(0)

CDRL: B005
29 October 1993

UNISYS

Library Development Handbook
Central Archive for Reusable Defense Software (CARDS)

Informal Technical Data

Central Archive for Reusable Defense Software

STARS-VC-B005/001/00
29 October 1993

CDRL: B005
29 October 1993

INFORMAL TECHNICAL REPORT
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Library Development Handbook
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B005/001/00
29 October 1993

Contract NO. F19628–93–C-0130
Line Item Informal Technical Data

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731–2816

Prepared by:

Strictly Business Computer Systems, Inc.
and
Electronic Warfare Associates, Inc.
under contract to
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Distribution Statement "A"
per Dod Directive 5230.24
Approved for public release, distribution is unlimited

CDRL: B005
29 October 1993

INFORMAL TECHNICAL REPORT For The SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Library Development Handbook
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B005/001/00
29 October 1993

Contract NO. F19628–93–C-0130
Line Item Informal Technical Data

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731–2816

Prepared by:

Strictly Business Computer Systems, Inc.
and
Electronic Warfare Associates, Inc.
under contract to
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

CDRL: B005
29 October 1993

Data ID: STARS-VC-B005/001/00

Distribution Statement "A"
per Dod Directive 5230.24
Approved for public release, distribution is unlimited

Developed by: Strictly Business Computer Systems, Inc. under contract to Electronic Warfare Associates, Inc.

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) Program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Schema (DoD Directive 5230.24) unless otherwise indicated by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0130, the STARS Program is supported by the military services with the U.S. Air Force as the executive contracting agent.

Permission to use, copy, modify, and comment on this document for purposes stated under Distribution "A" and without fee is hereby granted, providing that this notice appears in each whole or partial copy. This document retains Contractor indemnification to the Government regarding copyrights pursuant to the above referenced STARS contract. The Government disclaims all responsibility against liability, including costs and expenses for violation of property rights, or copyrights arising out of the creation or use of this document.

In addition, the Government, Unisys, and its subcontractors disclaim all warranties with regard to this document, including all implied warranties of merchantability and fitness, and in no event shall the Government, Unisys, or its subcontractor(s) be liable for any special, indirect, or consequential damages or any damages whatsoever resulting from the loss of use, data, or profits, whether in action of the contract, negligence, or other totious action, arising in connection with the use or perfomance of this document.

CDRL: B005
29 October 1993

INFORMAL TECHNICAL REPORT
Library Development Handbook
Central Archive for Reusable Defense Software
(CARDS)

Principal Author(s):

Brian Curfman *Date*

Steven Lewis *Date*

Jay Reddy *Date*

Approvals:

System Architect: *Kurt Wallnau* *Date*

Program Manager: *Lorraine Martin* *Date*

(Signatures on File)

CDRL: B005
29 October 1993

ABSTRACT

The process of developing a domain-oriented reuse library is elaborate. This Library Development Handbook provides an overview of the phases involved in developing such a library: domain analysis, library encoding, and library population. This Handbook presents a generic library population process that has been developed by the Central Archive for Reusable Defense Software library development team. This Handbook enumerates specific instructions and examples for populating a domain-oriented reuse library, based on this library population process. We are currently assessing the implications of legal issues on our policies and procedures. Appropriate risk reduction procedures will be detailed in subsequent releases.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 29 October 1993	3. REPORT TYPE AND DATES COVERED Informal Technical Report

4 TITLE AND SUBTITLE Library Development Handbook CARDS	5. FUNDING NUMBERS F19628-93-C-0130
6. AUTHOR(S) Brian Curfman, Steven Lewis, Jay Reddy	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Unisys Corporation 12010 Sunrise Valley Drive Reston, VA 22091	8. PERFORMING ORGANIZATION REPORT NUMBER STARS-VC-B005/001/00
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of the Air Force ESC/ENS Hanscom AFB, MA 01731-1816	10. SPONSORING/MONITORING AGENCY REPORT NUMBER B005

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution "A"	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The process of developing a domain-oriented reuse library is elaborate. This Library Development Handbook provides an overview of the phases involved in developing such a library: domain analysis, library encoding, and library population. This Handbook presents a generic library population process that has been developed by the Central Archive for Reusable Defense Software library development team. This Handbook enumerates specific instructions and examples for populating a domain-oriented reuse library, based on this library population process. We are currently assessing the implications of legal issues on our policies and procedures. Appropriate risk reduction procedures will be detailed in subsequent releases.

14. SUBJECT TERMS	15. NUMBER OF PAGES 36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

Table of Contents

1 Introduction....1

1.1 Purpose....1
1.2 Scope....1
1.2.1 Document Structure....1
1.2.2 Related Documents....2
1.3 Background....2

2 Library Development Process Overview....3

2.1 Roles in the Library Development Process....3
2.2 Library Development Process Description....4

3 Domain Analysis....6

3.1 Overview....6
3.2 Process Description....6
3.3 CARDS Domain Analysis Process....7
3.3.1 Domain Criteria....7
3.3.2 Products....14

4 Library Encoding....15

4.1 Overview....15
4.2 Process Description....15
4.3 CARDS Library Encoding Process....15
4.3.1 Encode the Domain Requirements....15
4.3.2 Encode the Generic Architectures....16
4.3.3 Encode the Traceability Matrix....16
4.3.4 Encode the Domain and Common Criteria....16
4.3.5 Products....16

5 Library Population....18

5.1 Overview....18
5.2 Process Description....18

5.2.1 Acquisition ... 18
5.2.2 Adaptation ... 19
5.2.3 Integration ... 20
5.3 CARDS Library Population Procedures ... 21
5.3.1 Acquisition ... 21
5.3.2 Identification ... 22
5.3.3 Screening ... 23
5.3.3.1 Domain Crit∂ria Evaluation ... 24
5.3.3.2 Common Criteria Evaluation ... 24
5.3.3.3 Adaptation Specification ... 25
5.3.3.4 Integration Potential ... 26
5.3.3.5 Library Inclusion Decision ... 26
5.3.4 Adaptation ... 27
5.3.4.1 Design ... 27
5.3.4.2 Implementation (Coding) ... 27
5.3.4.3 Stand-Alone Testing ... 28
5.3.4.4 Adaptation(s) Report ... 28
5.3.5 Integration ... 28
5.3.6 Integration Report ... 29
5.3.6.1 Products ... 29

6 Threaded Processes ... 31

6.1 Overview ... 31
6.2 SDF ... 31
6.2.1 Purpose ... 31
6.2.2 Policy ... 31
6.2.3 Procedures ... 31
6.2.3.1 SDF Organization and Contents ... 31
6.2.3.2 SDF Creation ... 32
6.2.3.3 SDF Updates ... 32
6.3 CM ... 32
6.3.1 Purpose ... 32
6.3.1.1 Definitions ... 32
6.3.2 Policy ... 33
6.3.3 Procedures ... 33
6.4 Configuration Management Request (CMR) Form ... 33
6.4.1 Purpose ... 33
6.4.2 Policy ... 33
6.4.3 Procedures ... 33
6.4.3.1 CMR Contents ... 33

6.4.3.2 CMR Process....34
6.5 Component Status Tracking....34
6.5.1 Purpose....35
6.5.2 Policy....35
6.5.3 Procedures....35
6.6 Library Development Quality Control....36
6.6.1 Purpose....36
6.6.2 Policy....36
6.6.3 Procedures....36

Appendices

Appendix A Component Status Sheet....A - 1

Purpose....A - 1
Instructions....A - 1

Appendix B Component Data Sheet....B - 1

Purpose....B - 1
Instructions....B - 1
SDF Identification....B - 2

Appendix C Domain Criteria Evaluation....C - 1

Purpose....C - 1
Instructions....C - 1
Domain Criteria Form....C - 1

Appendix D Common Criteria Evaluation....D - 1

Purpose....D - 1
Instructions....D - 1
Example Common Criteria Form....D - 1

Appendix E Component Evaluation Report....E - 1

Purpose....E - 1
Instructions....E - 1
Report Template....E - 1

Appendix F Software Change Request (SCR) Form....F - 1

Purpose....F - 1
Instructions....F - 1
Software Change Request Template....F - 3

Appendix G System Trouble Report (STR) Form....G - 1

Purpose....G - 1
Instructions....G - 1
System Trouble Report (STR) Template....G - 4

Appendix H Configuration Management Request (CMR) Form....H - 1

Purpose....H - 1
Instructions....H - 1

Appendix I CARDS Software Documentation Standard....I - 1

Purpose and Scope....I - 1
Applicability....I - 1
Responsibility....I - 1
Policy....I - 1
Background....I - 2
Standards....I - 2
Requirements Analysis Phase....I - 2
Design Phase....I - 3
Code and Debug Phase....I - 5
Integration Phase....I - 6
Maintenance Phase....I - 6
Related Policies....I - 7
Example Documentation Templates....I - 8
In-line Ada Documentation Templates....I - 8
In-line Script Documentation Template....I - 9
Version Description Document Template....I - 10

Appendix J Glossary....J - 1

Appendix K References....K - 1

1 Introduction

1.1 Purpose

This Library Development Handbook (LDH), describes the processes by which the contents of a domain-specific reuse library are generated, verified, and maintained.

The development of a domain-oriented reuse library encompasses aspects of domain analysis, domain engineering, library science, and application engineering. A domain-oriented reuse library provides the necessary infrastr ·ture for use of domain-engineering artifacts in building new applications in the domain. The library also provides a viable feedback mechanism for improving domain-analysis and domain-engineering processes.

1.2 Scope

This LDH specifies a process for developing the contents of a domain-specific reuse library and covers the following aspects of library development:

- Domain Analysis - See also Central Archive for Reusable Defense Software (CARDS) Technical Concepts Document (TCD) [5] for additional information.
- Library Encoding - See also CARDS Library Model Document (LMD) [4] for additional information.
- Acquisition, development/adaptation, and integration of software products for the library.
- Tools currently used by CARDS to support/enhance some key processes.
- Configuration management.
- Quality control of all products.
- Interfacing with library operations - See also Volumes I and II of the Library Operations Policies and Procedures Manual (LOPP) [8]. The LOPP describes the processes employed in the operation of an existing domain-oriented library. Volume I details the policies and procedures for an existing operational library. Volume II illustrates the day-to-day instructions implemented at the CARDS facility.

1.2.1 Document Structure

Every chapter in this document is logically delineated into two sections. The first section describes the overall processes or methods CARDS prescribes as required/desirable for any domain-oriented reuse libraries. This section also highlights the rationale for undertaking such processes.

The latter section describes the specific approach CARDS currently employs in support of the processes/methods described earlier. This section describes, in detail, the exact steps the CARDS library development team currently follows.

This delineation will help potential franchises [3] to choose the appropriate technologies and define steps for key processes based on their needs and resources. CARDS, like any other organization, is constrained by available technologies and resources in implementing every part of the process in developing domain-oriented reuse libraries.

As a final note, the concepts espoused by CARDS and other reuse programs are evolutionary. These concepts will be constantly refined/updated as more experience is accrued in building domain-oriented reuse libraries.

1.2.2 Related Documents

- CARDS TCD [5]
- CARDS Franchise Plan [3] - Describes initial library development and the library start-up activities.
- CARDS Acquisition Handbook [1] - Describes legal and contractual issues concerning reuse.
- CARDS Command Center Library Model Document [2] — Describes the CARDS Command Center Library model.
- CARDS LOPP [8]

1.3 Background

The charter of CARDS is to provide a "blueprint" for instituting domain-specific reuse. In support of this charter, CARDS is developing techniques for establishing domain-specific reuse libraries. The development of a domain-specific library is described as a set of general processes to accommodate specific methods a particular library may choose to employ.

Some of the library development processes, such as domain analysis, are immature. There are currently no universally accepted definitions for domain analysis and its products (such as domain model, generic architecture, etc.). This Handbook emphasizes the products a domain analysis process should generate without dictating the exact processes producing the products. The definitions of terms used within this document are located in the CARDS Glossary (See Appendix J).

2 Library Development Process Overview

2.1 Roles in the Library Development Process

The following roles must be filled during the library development process:

Component Engineer. The component engineer is responsible for evaluating components for the library, adapting the components if necessary, evaluating component criteria, analyzing the criteria, integrating components, and reporting the findings as appropriate. The component engineer is also responsible for developmental testing. However, independent testing is performed by the test engineer, who is a part of the library operations team.

Development Configuration Manager. The development configuration manager is responsible for overseeing proper configuration management (CM) of developmental hardware and software, library model, library contents, and accompanying documentation. The development configuration manager makes suggestions to the LCB and ensures decisions of the LCB are implemented. The development configuration manager is not responsible for configuration management of the operational library.

Library Control Board (LCB) Chair. The LCB chair is the point of contact for the LCB. The LCB is the deliberative body controlling the decision-making process for making recommendations about the contents of the library (model, components). The LCB also has approval authority for determining when to release new versions of a library to the Configuration Control Board (CCB) for customer release. The LCB is concerned with the technical issues related to library construction.

Domain Analyst. The domain analyst is skilled in domain analysis methodologies. The person supporting this role provides the procedural know-how on domain analysis. The domain analyst is responsible for defining the language, tools, and techniques used in performing the domain analysis. This person also documents the domain model and may be responsible for defining any generic architectures associated with the domain. This person is generally also responsible for training personnel in the use of domain analysis methodology, assessing conformance to any applicable standards and procedures, and providing expertise in the area of systems/software engineering.

Domain Expert. The domain expert has extensive knowledge of the application domain being examined. This person acts as a consultant to other domain analysis personnel, assists in defining model(s) of the domain, finding and analyzing existing systems, anticipating and assessing the impacts of future system requirements, and acts as a consultant to personnel building new application systems in the domain.

Knowledge Engineer. This person is responsible for modeling domains. The knowledge engineer works closely with the domain analyst and the domain expert in encoding the domain analysis products into a library model. The knowledge engineer is the point of contact for problems with the library model and documents changes to the library model.

Quality Assurance. Quality assurance of the products generated by the library development process is built into the process as part of Total Quality Management (TQM). The LCB ensures the quality of the products in the developmental library. The role of quality assurance is also to monitor the adherence of the library developers to the library development processes.

Configuration Control Board (CCB) - The authority responsible for CARDS configuration management. The CCB is not part of the library development process, however, it is the hand-off mechanism between the developmental and the operational library (See Figure 2-2).

2.2 Library Development Process Description

The library development process generates a library model and components supporting the model. The LCB reviews the library model and makes the recommendations to the CCB for operational release. On approval of the CCB, the model becomes part of the operational library. Refinements to the library model (operational or otherwise) are carried out as part of the library development process (See Figure 2-1).

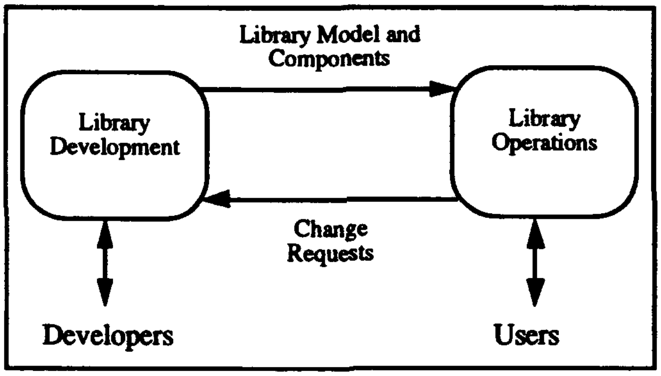


Figure 2-1 Library Development Process.

The library is developed in four distinct phases (See Figure 2-2). Each of these phases is described in detail in subsequent chapters.

1. **Domain Analysis.** Domain analysis is the process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain, based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain. Typically, this process generates products such as the domain model (specification of the problem space), generic architectures (specifications of the solution space), and a list of potential products that implement the generic architectures.

2. **Library Encoding.** Library encoding is the process of encoding the products of the domain analysis into a library model. The library model may contain additional information to aid the users in the search and retrieval of components from the library. Information such as the domain criteria specify the "form, fit, and function" a software product (component) should possess for inclusion into the library.

3. **Library Population.** Library population is the process of acquiring/developing components to be incorporated into the reuse library. The components provide users with specific implementations of the generic architecture encoded in the library model.

4. **LCB and CCB Approval.** The products of library development (library model and components) have to be approved by the LCB and CCB before they become part of the operational library. Library Development Phases.

3 Domain Analysis

3.1 Overview

Domain analysis is the process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain, based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

Current literature defines the scope of domain analysis to varying degrees. Some of the literature defines domain analysis to encompass all of the activities of domain engineering (analysis, design, and implementation). Others define domain analysis to encompass defining the "problem-space" and proposing a "solution-space" (skipping implementation). Yet others define domain analysis as an activity concerned only with the definition of the "problem-space" for a domain. Some methods restrict the "problem-space" to requirements only and others include all the information that can be derived from existing systems (architectures, design rationale, tradeoffs, etc.). The definition used by CARDS deals exclusively with defining the "problem-space" of a domain and the "problem-space" is not restrictive to domain constraints only. Such a definition clearly recognizes the importance of all information, not just requirements, that can be derived from existing systems. This definition of domain analysis also delineates the "problem-space" from the "solution-space" of a domain. Such a distinction supports the concept of multiple stakeholders for a given domain. For example, a Government agency may retain control of "problem-space" of a domain and let independent contractors (or another agency) propose and/or implement the "solution-space".

3.2 Process Description

The concept of domain analysis is relatively new; hence the existing process descriptions are immature. There are several domain analysis processes [7][8][11] in existence that may be appropriate for specific domains. No general agreement on the processes and products of domain analysis currently exists.

The following is a sketch of the domain analysis process (the following steps are not necessarily sequential):

- Capture the domain constraints through reverse engineering, domain modeling, and prototyping.
- Design a generic architecture of a system in the domain; several techniques could be utilized including object-oriented analysis/design and reverse engineering of existing systems.
- Allocate domain constraints to subsystems in the generic architecture. Each subsystem in the generic architecture is then bound by a set of subsystem requirements.

- Derive domain criteria used to evaluate potential products for inclusion into the reuse library.

A possible by-product of domain analysis is a list of identified components that (partially) satisfy each subsystem within the generic architecture. The identified components will aid greatly in the creation of the domain criteria.

3.3 CARDS Domain Analysis Process

To date, CARDS has not undertaken significant domain analysis efforts. However, we are in the process of conducting analysis of several domains, and as our knowledge matures, we will update the appropriate techniques. A large portion of the domain analysis process ("problem space" and "solution space") for the Command Center domain was completed by Portable, Reusable Integrated Software Modules (PRISM). Using this as a basis, CARDS initiated the process of developing domain criteria as detailed below.

3.3.1 Domain Criteria

Domain criteria are a composite of three sets of requirements:

- Class requirements.
- Architectural constraints.
- Implementation requirements.

Domain criteria need to be developed to measure a component's applicability to the domain and how well the component's functionality meets the domain constraints. A component successfully meeting the domain criteria yields a higher level of confidence for users of the systems created with that component.

The determination of domain criteria is an iterative process involving the analysis of domain and subsystem requirements and disintegration of the existing systems. The process of defining domain criteria (See Figure 3-1) is described as follows:

1. **Identify a subsystem in the generic architecture** - the first step in creating domain criteria is to identify a subsystem in the generic architecture.

2. **Define class(es)** - based on the products of domain analysis (domain constraints and subsystem requirements), define a component class satisfying the identified subsystem (e.g., *database management systems* is a component class that can fully satisfy a *database management systems* subsystem; *spreadsheets* is a component class that can fully satisfy a *briefing systems* subsystem) (See Figure 3-2). If one component class cannot fully satisfy the identified subsystem, the subsystem then needs to be

further decomposed into sub-functions until each sub-function can be fully satisfied by a component class (e.g., *email* and *word processing* classes each partially satisfy an *office automation* subsystem; therefore, *email* and *word processing* become sub-functions fully satisfied by the *email* and *word processing* classes, respectively) (See Figure 3-3). In the latter case, each sub-function then becomes bound by a set of sub-function requirements. If the domain is immature, it is possible that no component class exists that can fully satisfy the subsystem and that the subsystem cannot be logically or functionally decomposed into sub-functions. In the case of library development, the library development team may choose to not populate the subsystem or in the case of systems development, the systems development team may choose to custom develop a component that will fully satisfy the subsystem. Note that the same component class may satisfy more than one subsystem in the generic architecture. Also note in the examples given above, a component class may or may not be represented by the same name as the subsystem. Either case is acceptable.

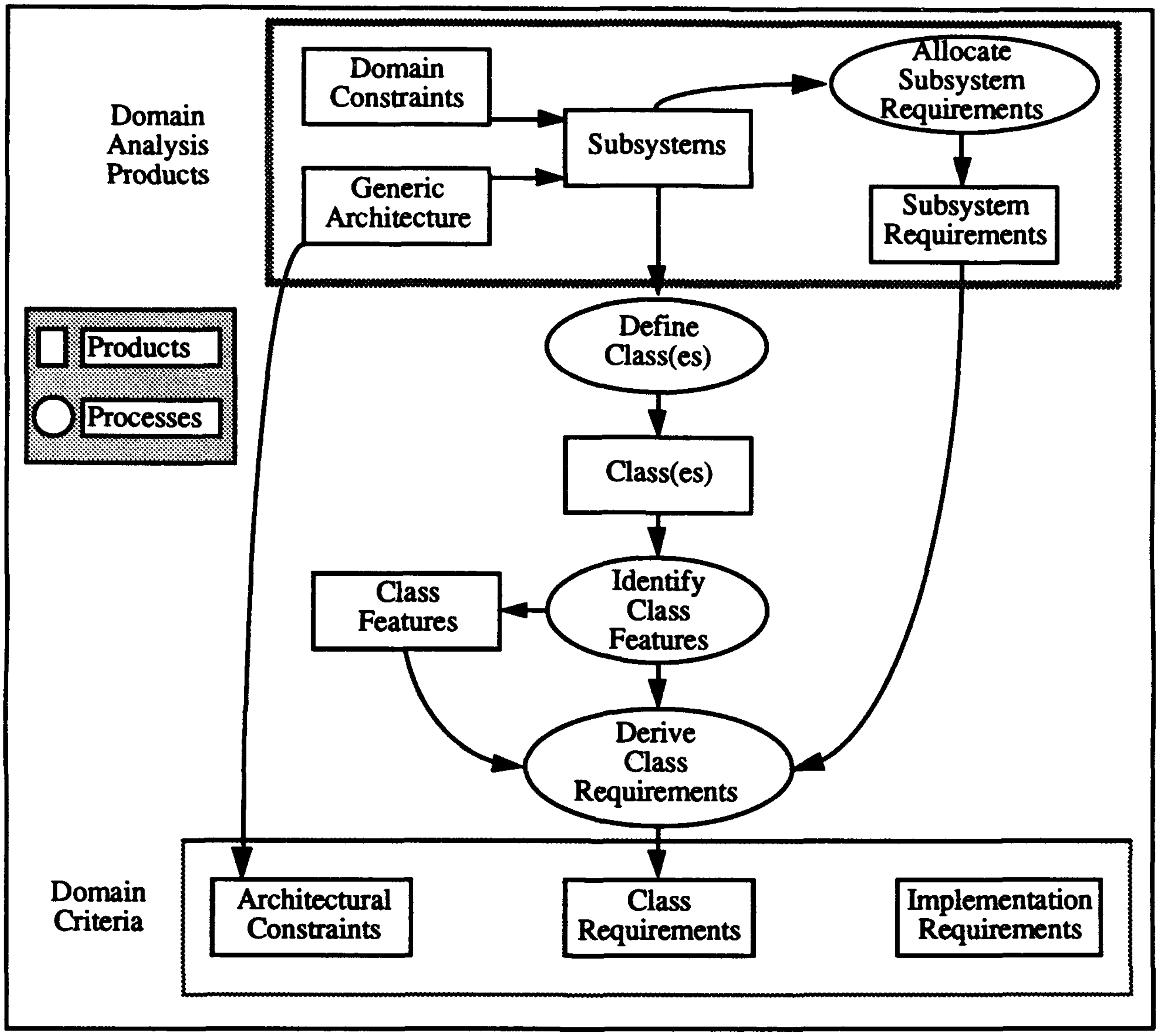


Figure 3-1 Domain Criteria Creation Processes

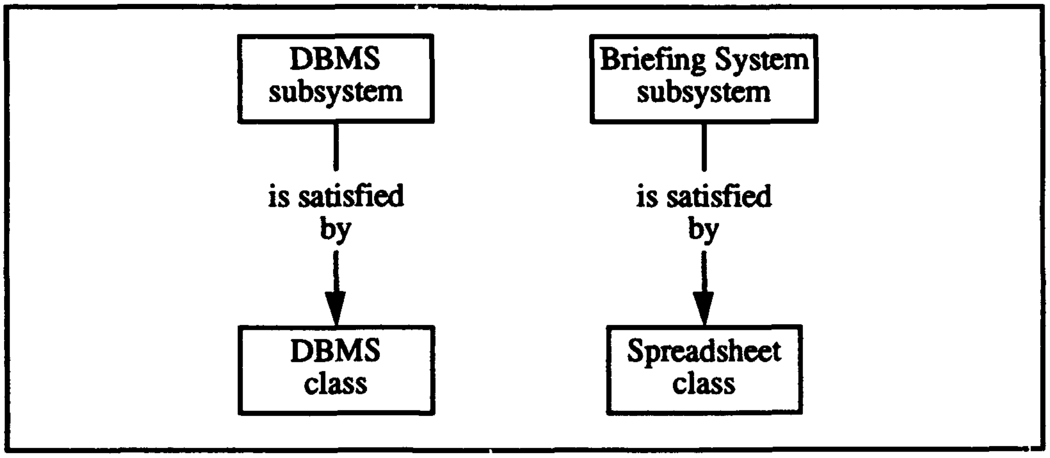


Figure 3-2 A Subsystem Fully Satisfied By One Class

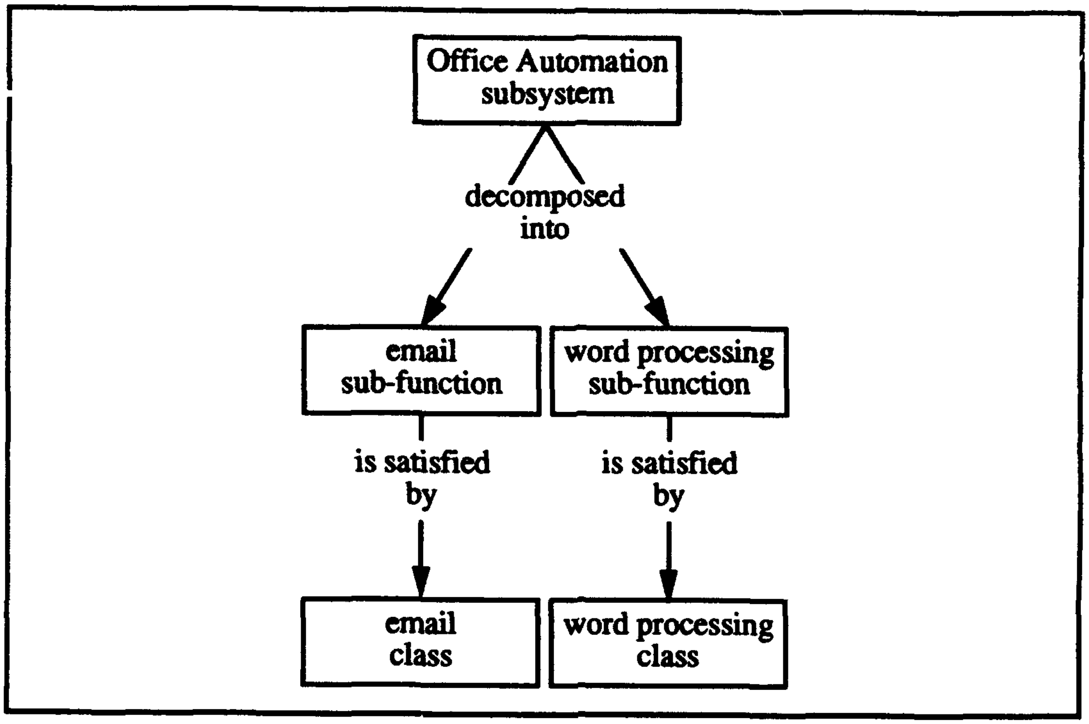


Figure 3-3 A Subsystem Satisfied By More Than One Class

3. **Identify class features** - various techniques can be employed to identify class features, which are the general characteristics and functionalities for a given class of products/components described in terms inherent to the class. Some methods include performing dis-integration of existing systems/subsystems, analyzing trade periodicals, and knowledge acquisition from class experts. Class features are not dependent upon any domain. They may be used across domains, and in the case of a class satisfying more than one subsystem in the generic architecture (See Figure 3-4), they can also be used across subsystems. The class features list is continuously updated as new systems/subsystems are analyzed. The list of class features aid in the development of class requirements. Examples of spreadsheet class features could include: application will perform automatic calculations of data, application provides bar chart capabilities, and application provides scientific numeric formatting.

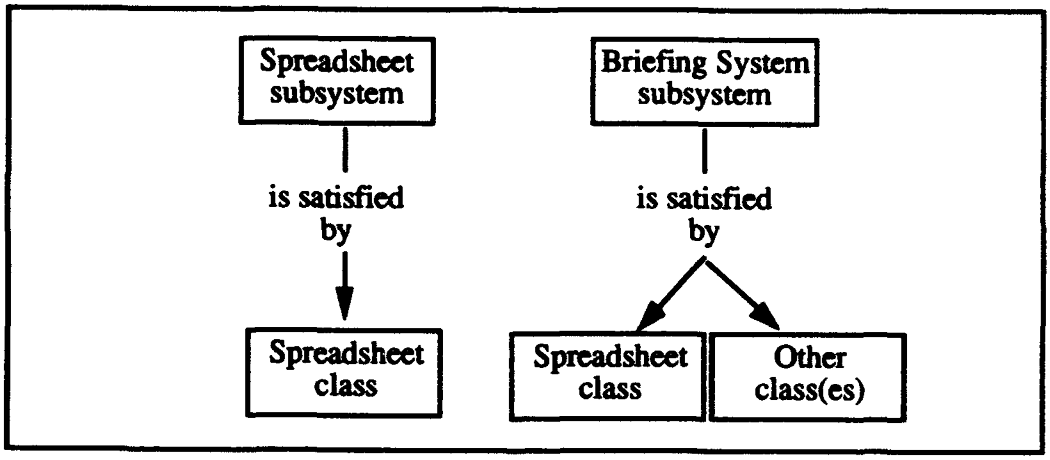


Figure 3-4 Two Subsystems Fully Satisfied By One Class

4. **Derive class requirements** - class requirements represent the functions that a component must possess to satisfy the subsystem requirements and class features. There are two types of subsystem requirements: low-level, those phrased in class terminology, and high-level, those phrased in domain terminology. Class requirements are identified in two ways (See Figure 3–5):

   A. Performing a direct allocation of the low-level subsystem requirements to class requirements (See Figure 3-6). This is done by using the low-level subsystem requirements word for word as class requirements.

   B. Qualifying the high-level subsystem requirements with class features (See Figure 3-7) by:

      1. mapping the high-level subsystem requirements to corresponding class features and

      2. performing an allocation of the subsystem requirements along with the qualifying class features to class requirements.

In the example given (See Figure 3-7), the subsystem requirement application will create formal documents versed in domain terminology. In class terminology, the definition of a formal document is the ability to provide headers and footers within a document; therefore, the class features application provides header capabilities and application provides footer capabilities are used to qualify the subsystem requirement.

Sources for class requirements include the analysis of existing products, published requirements, domain expertise, and the library model. Any class requirements that include domain terminology must be qualified with class features. This feature benefits the individual not versed in the terminology of the domain. When class requirements become part of the domain criteria, they are phrased in an interrogative manner so a response can be given.

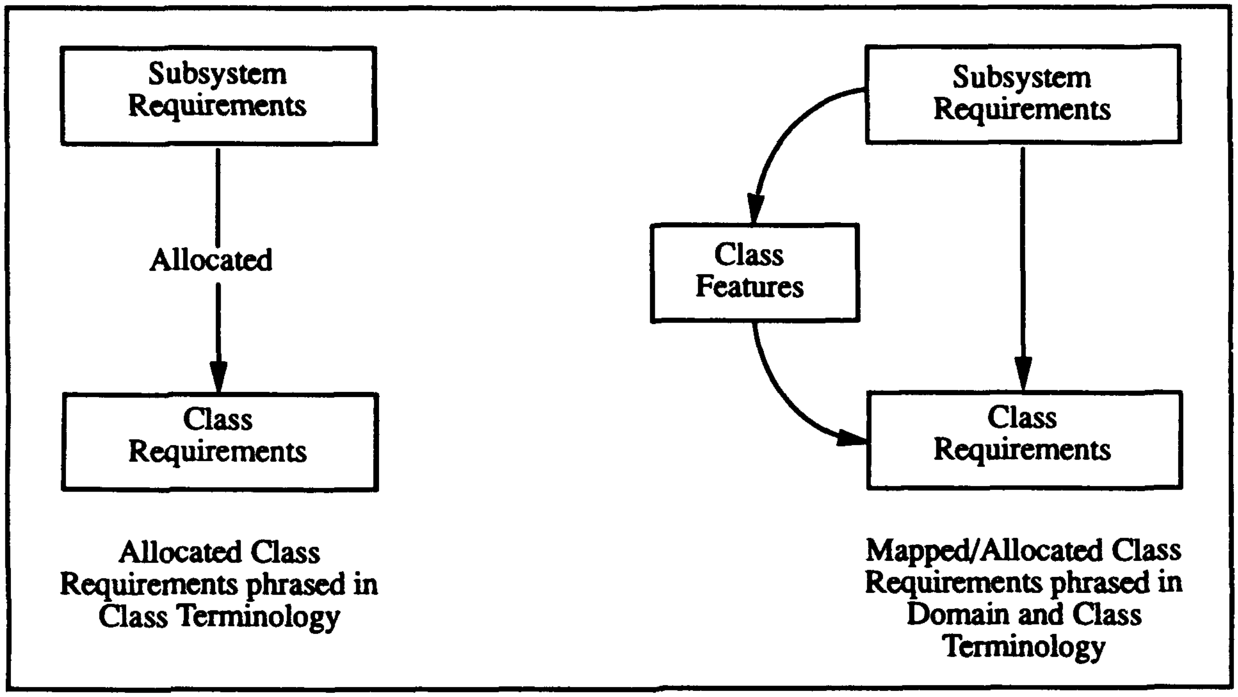

Subsystem Requirements
Allocated
Class Requirements
Allocated Class Requirements phrased in Class Terminology
Subsystem Requirements
Class Features
Class Requirements
Mapped/Allocated Class Requirements phrased in Domain and Class Terminology

Figure 3-5 Class Requirements Identification

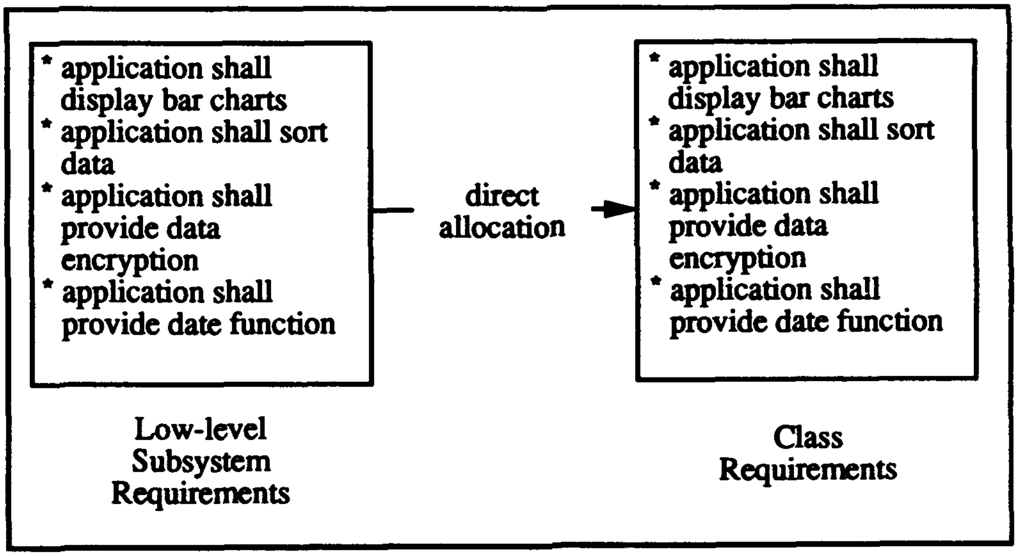

* application shall display bar charts
* application shall sort data
* application shall provide data encryption
* application shall provide date function
direct allocation
* application shall display bar charts
* application shall sort data
* application shall provide data encryption
* application shall provide date function
Low-level Subsystem Requirements
Class Requirements

Figure 3-6 Allocation Of Subsystem Requirements To Class Requirements

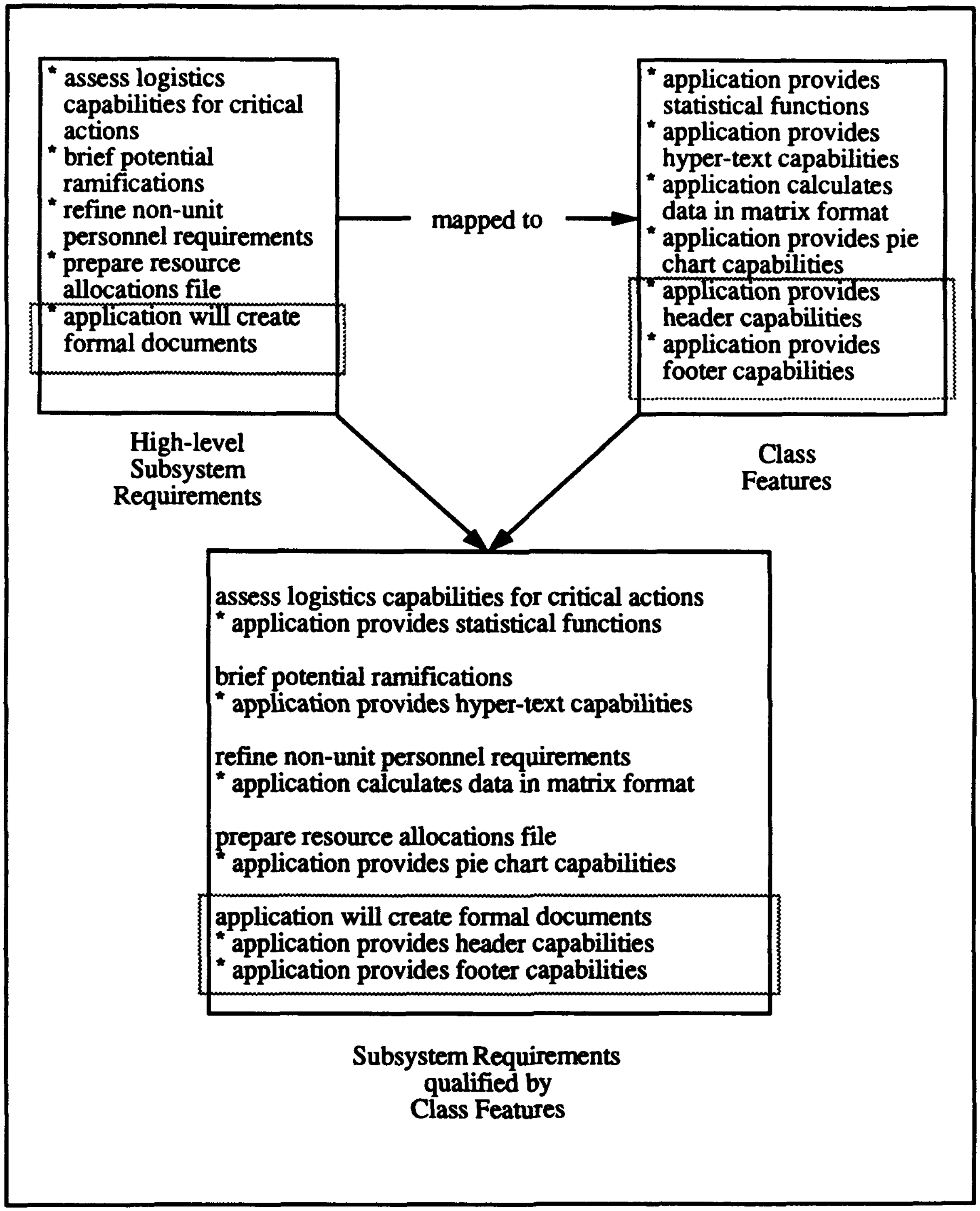

* assess logistics capabilities for critical actions
* brief potential ramifications
* refine non-unit personnel requirements
* prepare resource allocations file
* application will create formal documents
mapped to
* application provides statistical functions
* application provides hyper-text capabilities
* application calculates data in matrix format
* application provides pie chart capabilities
* application provides header capabilities
* application provides footer capabilities
High-level Subsystem Requirements
Class Features
assess logistics capabilities for critical actions
* application provides statistical functions
brief potential ramifications
* application provides hyper-text capabilities
refine non-unit personnel requirements
* application calculates data in matrix format
prepare resource allocations file
* application provides pie chart capabilities
application will create formal documents
* application provides header capabilities
* application provides footer capabilities
Subsystem Requirements qualified by Class Features

Figure 3-7 Qualification Of Class Requirements By Class Features

5. **Identify architectural constraints** - These are the relations/interfaces between subsystems within the generic architecture. Sources for determining these constraints include published requirements, domain expertise, existing systems, and the library model.

6. **Identify implementation requirements** - These are the constraints that have to be satisfied by the products that are composed into specific systems. Example constraints are hardware platforms, operating system versions, environments, etc. Typically these constraints are gathered from existing systems.

7. **Determine which constraints are critical to the domain.**

3.3.2 Products

The following is a list of potentially required and/or desirable products of domain analysis and is in no way comprehensive. Certain domains and applications may demand additional products as a result of domain analysis.

- Domain Constraints
- Technical Reference Model
- Generic Architecture
- Allocation and Traceability Matrix
- Identified Component List
- Domain Criteria

4 Library Encoding

4.1 Overview

The library encoding process encodes the products of the domain analysis into a library model. The library model organizes and presents the products of domain analysis in a manner that facilitates access to the products by the library users.

System composition tools make use of the encoded constraints in generating valid combinations of systems/sub-systems using the existing components in the library. In addition to the products of domain analysis, the library model may contain additional information to assist the library users in the search and retrieval of components. Information such as domain criteria collected on specific components is also encoded in the library model.

4.2 Process Description

The following are the steps of library encoding:

1. Encode the domain requirements.
2. Encode the generic architectures.
3. Encode the traceability matrix.
4. Encode the domain and common criteria.

The process will be further refined as the CARDS processes mature.

4.3 CARDS Library Encoding Process

See the CARDS Command Center Library Model Document [4] for examples on encoding:

- Domain Requirements,
- Generic Architectures,
- Traceability Matrix, and
- Domain and Common Criteria.

4.3.1 Encode the Domain Requirements

Domain requirements may be depicted in several views to define the context and behavior of the systems in the domain. The customary views represented are [10]:

- A context model describing the interactions between the domain and its interfaces to external domains.
- A behavioral model illustrating roles, objects, data items, and relationships.
- A logical model illustrating structure, object characteristics, and interactions.

It is important to maintain consistency between several views of the requirements. Additionally, these views should map to elements of the generic software architecture.

4.3.2 Encode the Generic Architectures

A domain architecture is depicted in a physical view of the domain representing a set of required domain components or subsystems with an associated set of relationships between components. Communications channels should be an explicit part of the architecture [7].

The encoded architecture provides the necessary context and constraints to a system composition tool. Such a tool would use the architectural constraints (along with implementation constraints) imposed by the generic architectures in generating valid configurations of architectures. CARDS is creating a technical document on system composition.

4.3.3 Encode the Traceability Matrix

Traceability between domain requirements and generic architecture(s) facilitates rapid prototyping efforts within a domain-specific reuse library. Library users may drive system composition by selecting domain requirements, derive specific requirements by selecting subsystems/components of an architecture, or a combination of both. Traceability aids in trade-off analysis of both requirements and system-specific architectures. Additionally, traceability enhances library maintenance activities.

4.3.4 Encode the Domain and Common Criteria

Domain and common criteria specify the "form, fit and function" a component should possess to qualify as a reusable asset within the domain/library. Formally encoded domain criteria can be used by the system composition tools in producing configurations conforming to user needs. Domain criteria can be used as discriminators by the library users in the selection of a component from a field of several candidates. Clear definition of the criteria may enable (partial) automation of the component qualification process.

Thus, the process of encoding domain criteria should facilitate system composition capabilities, account for efficient search and retrieval of components, and lay the foundation for automated qualification tools.

4.3.5 Products

The products of library encoding are:

1. Encoded domain requirements
2. Encoded generic architecture
3. Encoded traceability matrix
4. Encoded domain criteria

5 Library Population

5.1 Overview

The library population process is needed to qualify and incorporate software products into the reuse library. A software product can consist of:

- A full set of life-cycle documentation, code, and test suites.
- User/programmer documentation and executable code.
- Any logical subset of life-cycle documentation and code, e.g., a software product could just consist of a software requirements specification and no code.

In addition to the generic library population process (Section 5.2), which can be applied to any library, this chapter covers the CARDS implementation (Section 5.3).

5.2 Process Description

Products are qualified by being passed through the population process which includes three phases: acquisition, adaptation, and integration (See Figure 5-1).

5.2.1 Acquisition

During the acquisition phase, a class is chosen to be populated. Software classes are derived from the domain architecture during domain analysis. Generally, there is a one-to-one mapping between a subsystem and a class fulfilling the functional requirements of the subsystem, e.g., the desktop publishing class satisfies the functional requirements of the desktop publishing subsystem. In addition to this one-to-one mapping, there may be existing software products which would immediately fulfill the requirements of the specified class. If a one-to-one mapping is not possible, a subsystem may require the functional capabilities of more than one class (See Figure 3-3). Note that it is also possible that one class may satisfy the functional requirements of more than one subsystem (or at least some of the functional requirements of more than one subsystem) (See Figure 3-3). In addition, a software product may fulfill the requirements of more than one class.

After selecting a class, potential software products are identified, screened and evaluated for inclusion in the domain-specific reuse library. Identification of products entails gathering information on each potential product. This information is passed on to the screening process for further evaluation. During the screening process, the list of potential products is prioritized, enabling labor intensive evaluations to be performed with a higher acceptance rate, since less qualified components are given a low priority.

After the products pass the screening process they become candidate components for the evaluation process. The candidates showing the most potential for the library are then evaluated with respect to the domain and common criteria. The domain criteria are used to evaluate the product's "form-fit-function" relative to the domain architecture. The domain criteria provide information regarding class requirements, architectural requirements, and implementation requirements. The common criteria are used to help library users assess a component's compatibility with the user's configuration and needs. The common criteria provide information such as licensing, product/technical support, and hardware/software requirements. Based on the results of the domain and common criteria evaluation, a recommendation is provided to the LCB regarding thc inclusion of the product into the library.

The LCB is the control body making the decisions concerning the library components, e.g., which components to acquire, reject, or modify. The LCB is concerned with the technical issues related to library construction.

Accepted products are then integrated into the library as reusable assets. Information pertinent to products not accepted by the LCB is archived for possible future retrieval. Any required adaptations to the product are specified at this stage and passed to the adaptation phase. However, if no adaptations are required, the process moves on to the integration phase.

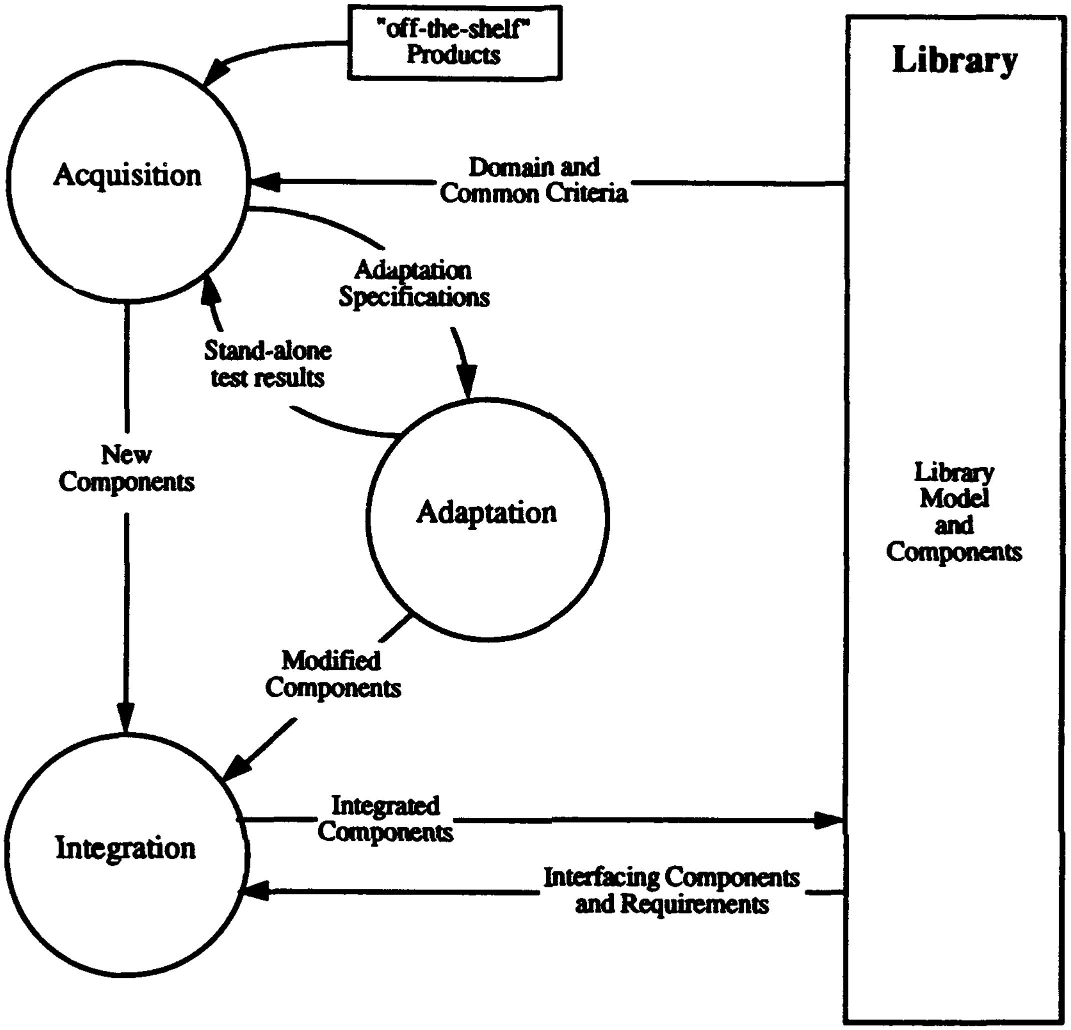


Figure 5-1 Library Population Process

5.2.2 Adaptation

A component may require modifications or enhancements to qualify for the reuse library. When this situation arises, the component must pass through the adaptation phase. The three steps of the adaptation phase are design, implementation, and stand-alone testing. Each step is documented and kept with the other information pertinent to the product being adapted. The design step involves engineering an implementation approach to the required adaptation. Design reviews should be employed according to standard software development techniques. Next, implement the design. The code should be reviewed as needed to ensure adherence to the design specifications and any pertinent standards. The next step in the process is stand-alone testing.

Stand-alone testing is conducted to verify that the software adaptations meet the functional, performance, and interface requirements set forth by the component. All test plans, procedures, and results should be formally documented and under configuration management control. The results of these activities are fed back to the acquisition phase.

At this time no formal process has been created for the adaptation of a non-code component, i.e., document. However, CARDS foresees the need for such a process. For example, a document may contain all the information required of it, but may not be in the proper format or conform to a particular standard, i.e., Standard Generalized Markup Language (SGML).

5.2.3 Integration

The integration phase verifies that a software product meets the architectural constraints imposed by the generic domain architecture. All products in the library, regardless of their origin (e.g., off-the-shelf, modified off-the-shelf, and wrappers), are integrated with other components in the library. Even non-code components (i.e., document) may be required to be in a specific format to work with other components as defined by the components architecture. However, it may not be necessary for all components to be integrated.

The integration of components is a critical phase in the library population process. The integration of a component with the other components in the library provides the library users with the ability to extract whole subsystems of components with which to design their library rather than having to build their system with individual component pieces. The integration of components may entail modifying the library model to accommodate the new component.

5.3 CARDS Library Population Procedures

The following process details the initial work necessary in populating the library. If the library is already populated with some components, then appropriate modifications to this process are necessary.

5.3.1 Acquisition

During the acquisition phase, potential software products are identified, screened and then evaluated for inclusion in the CARDS domain-specific reuse library. Products accepted by the LCB are then integrated into the library. The following procedure provides a structured approach to the acquisition of reusable software products:

1. Select a class to populate. A class is considered a specific software category whose members are similar in function and purpose.

2. Perform a market survey to identify potential products and compile information from vendors. Products can be commercial off-the-shelf (OTS), government OTS, public domain, etc., as previously defined

3. Generate a list of potential products.

4. Screen the list of potential products to narrow the field to a selected set of products based on key factors, e.g., with which platforms the product is compatible, availability of demonstration software or product software itself, etc.

5. Perform an evaluation of each product using class-specific domain criteria and common criteria.

6. Specify any adaptations/enhancements to the procured product necessary to fully satisfy the architectural requirements defined by the library model.

7. Specify the integration potential of each procured product.

8. Make recommendations to the LCB concerning the inclusion of each evaluated product into the library.

The following sections provide a detailed, procedural narrative of the acquisition process. Throughout this process, data is collected and organized on a set of forms and controlled within a Software Development Folder (See Section 6.2). Table 5-1 identifies each form and the step(s) where each is created or updated.

Table 5-1 Acquisition Steps and Related Forms.

Acquisition Step	Forms Created	Forms Updated
Product Identification	Component Data Sheet	
Screening	Component Status Sheet Software Development Folder Class Screening Report	Component Data Sheet
Domain Criteria Evaluation	Domain Criteria Form Component Evaluation Report	Component Status Sheet Software Development Folder
Common Criteria Evaluation	Common Criteria Form	Component Status Sheet Component Evaluation Report Software Development Folder
Adaptation Specification		Component Status Sheet Component Evaluation Report Software Development Folder
Integration Potential		Component Evaluation Report
Library Inclusion Decision		Component Status Sheet Component Evaluation Report Software Development Folder

5.3.2 Identification

During product identification, information on products suited to the domain is compiled. The following procedure defines the CARDS identification process:

1. Select a class to populate.
2. Generate a list of potential software products that will fully or partially meet the requirements of the class, via brainstorming, analyzing trade periodicals, and reviewing the domain analysis products, e.g., a list of requirements for the systems within the domain a generic architecture, a list of potential products, and domain criteria for subsystems.
3. Compile the information relevant to the Component Data Sheet (See Appendix B) by contacting sales representatives (if COTS) or support organizations (if GOTS) for each product.
4. Complete a Component Data Sheet for each product. The Component Data Sheet is utilized to collect general information and is useful in screening out incompatible components early in the evaluation process.
5. Obtain product information from sources other than the vendor (e.g., marketing literature, product reviews, etc.); also, obtain demonstration software, if available.
6. Pass the product information to the screening step.

5.3.3 Screening

During the screening step, the list of potential products is prioritized. This enables labor intensive evaluations to be performed with a higher acceptance rate, since less qualified components are given a low priority. The following procedure provides a structured approach to the screening step:

1. Rank the identified products relative to their compatibility with the selected domain.
2. For each potential product for incorporation into the library:

   A. Create a Software Development Folder (SDF). The SDF acts as a repository for all forms related to the product.

   B. Create the Component Status Sheet (See Appendix A). The Component Status Sheet is used to track and control a component throughout the library population process.

C. Place the Component Data Sheet and the Component Status Sheet in the SDF.

3. Archive all information pertaining to products which were not recommended for further evaluation. Note that currently, there has been no formal procedure developed for archiving information. All information to be archived has been placed in compressed format on the CARDS network. This information can be made available to the library users. Future changes in the product or the domain status may warrant reassessment of previously discarded products.

4. Generate a list of archived information. This is done so that there is a reference to all information previously gathered by the evaluation team.

5. Generate a Class Screening Report including a list of all components screened and their current status. This is to be presented to the LCB.

5.3.3.1 Domain Criteria Evaluation

Domain criteria are used to evaluate components based on class requirements, architectural requirements, and implementation requirements. The domain criteria establish the component's "form-fit-function" relative to the domain architecture. "Form" refers to the usability of the component for the domain. "Fit" refers to the compatibility with other components already in the library. "Function" refers to the functionality the component exhibits when evaluated against the class-specific domain criteria. The following procedure provides a structured approach to completing the domain criteria evaluation:

1. Evaluate the component based on the domain criteria questions found on the Domain Criteria Form (See Appendix C). The evaluation may require obtaining a demonstration version of the software, having a demonstration given by a sales representative, or obtaining additional information through the vendor.

2. Add the Domain Criteria Form to the SDF of the product.

3. Create the Component Evaluation Report (See Appendix E) and report on the domain criteria evaluation. This evaluation report summarizes a product's "form-fit-function" within the domain.

4. Add the Component Evaluation Report to the SDF of the product.

5. Update the Component Status Sheet.

If the evaluation raises questions concerning the validity of the domain criteria or other aspects of the library model, the evaluation may be suspended until all concerns are resolved.

Changes to the domain criteria should be effected through a formal process and all forms should be placed under configuration management (See Section 6.3). In the event the LCB deems the unfulfilled criteria critical enough to deny product acceptance, all information compiled on the product should be archived. This archived information can be used in case changes to the domain or product warrant reassessment.

5.3.3.2 Common Criteria Evaluation

Common criteria are domain-independent criteria used to evaluate components regardless of the class. The common criteria help library users assess a component's compatibility with their particular configuration and needs. The common criteria provide information on licensing, demonstration software availability, product support facilities, etc., and can be used to determine the product's reliability, maintainability, portability, and security. The following procedure provides a structured approach to completing the common criteria evaluation:

1. Evaluate the component based on the common criteria questions found on the Common Criteria Form (See Appendix D).

2. Add the Common Criteria Form to the SDF of the product.

3. Update the Component Evaluation Report by providing a summary of the common criteria evaluation. Note that the common criteria are not normally used to decide whether or not the component should be included in the library.

4. Update the Component Status Sheet for the product.

5.3.3.3 Adaptation Specification

Any enhancements/modifications necessary for the inclusion of a product in the library should be specified by the product evaluator(s). The adaptation specification(s) are documented in accordance with the specified software documentation standards (See Appendix I) and should account for:

- Functional requirements,

- Performance requirements,

- Interface requirements,

- Design constraints.

Enhancements in the form of stand-alone software products (a.k.a. wrappers) are considered new components. A new SDF is created for each new component and is cross referenced to the product being adapted. The adaptation specification is placed into the wrapper's SDF along with a Component Data Sheet and Component Status Sheet.

When the LCB decides to accept a product with adaptations, the adaptations to the component are effected and the modified component undergoes stand-alone testing (See Section 5.3.4.3). The following provides a structured approach to the adaptation specification procedure:

1. Specify any adaptations necessary for the component to be accepted into the library.
2. Update the Component Evaluation Report with the adaptations.
3. Update the SDF of the product.
4. Update the Component Status Sheet for the product.
5. Pass the adaptation specification(s) on to the integration potential step.

5.3.3.4 Integration Potential

The integration potential of a component is assessed during the domain criteria evaluation. Any information regarding the product's ability to interface with other products should be noted in the evaluation report.

5.3.3.5 Library Inclusion Decision

Evaluation of the domain and common criteria results in a recommendation to the LCB. Possible recommendations are:

- Include the product "as is".
- Do not include the product.
- Include the product with adaptations (an estimate of the cost and resources for the adaptation should be provided).

The recommendation and supporting rationales are included in the Component Evaluation Report which is presented to the LCB. The evaluation report should also include:

- An executive summary.
- Details on where the component fits into the architecture,.
- Any miscellaneous notes regarding the evaluation not already mentioned in the domain criteria and common criteria sections, e.g., concerns or compliments regarding technical support, installation complications, etc.
- All the associated forms.

The Component Status Sheet is updated to note that the product is now under consideration for procurement.

The LCB decides whether or not to include the product in the library. If it is decided that the product should be included in the library, the product may have to be purchased. The product, however, does not have to physically be in the library. The library may only contain a reference to the product and information on how it can be procured. The decision of the LCB is recorded on the Component Status Sheet. If the LCB decides not to include the product in the library, then the reasons for denial are noted and placed in the SDF and the information regarding the product is archived.

5.3.4 Adaptation

In the adaptation phase, existing software products are modified or enhancements are developed as new software products. Software products are designed, implemented (coded) and then tested in a stand-alone configuration. The actual methods and techniques employed are at the discretion of the individual developer or dictated by library-specific software development standards. The results of these activities are fed back to the acquisition phase.

5.3.4.1 Design

The purpose of the design phase is to engineer an implementation approach to satisfy the levied performance, functional, and interface requirements within the specified implementation constraints. For new products developed for the library, various design methodologies (e.g., functional decomposition, object-oriented, and layered virtual machines) can be employed. Regardless of technique, the design is documented in accordance with the established software documentation standards. The documentation produced includes:

1. A description of the design including both an architectural view and a functional view.

2. Interface design descriptions including interfaces to hardware, other software, and the user, as applicable.

3. A program data design description.

When modifying existing software products, a significant portion of the design activity may actually be an analysis of the current design to determine the best way to make the specified modifications.

For both new and modified products, one or more design reviews should be held. The number and timing of the reviews depends upon the size and complexity of the design. The purpose of the reviews is to obtain consensus on the design and to verify the design will indeed meet all of its requirements. A typical outcome of a design review is a list of action items which must be closed before continuing with the design or passing the design on for coding. Any documentation produced should be included in the SDF.

5.3.4.2 Implementation (Coding)

The purpose of the implementation phase is to realize the design specifications in a machine-executable form.

During implementation, source code is generated fulfilling the intent of the design. Usually a top-down or bottom-up approach is taken to coding the design. With either approach, a portion of the code, a unit, is generated and then debugged using drivers, stubs, and pieces of previously debugged units. In a top-down fashion, the higher level units are implemented first with stubs provided to satisfy calls to lower level units. With the bottom-up approach, the lowest level units are prepared first and drivers are used to exercise them. Debugging is a vital task to ensure the maturity of the code prior to being formally tested.

The library developers should agree on a single high order language for new development and major modifications. This will simplify the software development process and limit the size of the support software environment.

All code produced should be reviewed for adherence to the design specification and coding standards. These reviews ensure that quality control is built into the process and represent another method to improve the quality of the code prior to testing.

Other products of the implementation phase include information pertaining to the process of preparing the source code for execution and an exact description of the constituent files of the product. This information should be documented and retained as specified in the software documentation standards (See Appendix I).

5.3.4.3 Stand-Alone Testing

Stand-alone testing is conducted to verify that the software adaptations meet their functional, performance, and interface requirements in a stand-alone configuration.

The software is to be subjected to a series of tests verifying its correctness and integrity. Tests should be designed to isolate the component and execute this stand-alone configuration.

The test plans, procedures, and results should be formally documented and controlled. Not only do the tests substantiate the verification of the software but they can also be an important tool during software maintenance. If the tests are documented and controlled, then they can be easily updated and repeated to verify the integrity of future product modifications.

Upon satisfactory performance in a stand-alone configuration, the component passes into the integration phase of the library population process. If the component fails the stand-alone testing, then the LCB is informed of the details of the failure and all pertinent information is archived.

5.3.4.4 Adaptation(s) Report

Results of the adaptation phase are to be summarized in an adaptation report. This report will include information regarding all steps of the adaptation process from adaptation specification to stand-alone testing. Note that at this time, there is no formal report template used by CARDS to implement this procedure. For now, all documentation is to be included in the component's SDF.

5.3.5 Integration

The integration phase verifies tha. a software component meets the architectural constraints imposed by the generic architecture. Note that the evaluation of the components against domain criteria (in the acquisition phase) determines the likelihood of integration of the component. All components in the library, regardless of their origin (i.e., off-the-shelf, modified off-the-shelf, and wrappers), need to be integrated into the library.

The library model defines the subsystems and classes of components. For every software product acquired by the library, the component evaluator files a System Change Request (SCR) form (See Appendix F) to the configuration management team which encodes the component into the library model (See Section 4.2). The constituents of the SDF associated with the component are placed at the component node in the library. The SDF contains electronic versions of the component itself and associated forms (data sheet, status sheet, evaluation report, documentation, etc.). References to all non-electronic data associated with the component are also placed at the node.

Once the component is inserted into the library model, it must be integrated with the existing components in the library. Integration of components within a library defines subsystem functionality within an architecture for prototype and demonstration purposes. Since the integration of components may require significant resources, management approval of the scope and extent of integration should be obtained. For every integration effort, a plan is prepared and presented to management for approval. The plan details the level of integration that must occur between the existing products and the product being integrated. Issues contained in the integration plan include:

- The level of integration; this may be either (1) specification level (integrating with non-executable components) or (2) execution level (integrating with executable products).
- Required changes to the library model due to component incompatibilities.
- Additional wrapper(s) needed.

Upon the approval of management, the integration is performed according to the plan.

5.3.6 Integration Report

At the time of this writing, there is no CARDS-specific procedure defined for an integration report.

5.3.6.1 Products

The products of integration are:

- Enhanced library model
- Documented integration plans
- Integration report
- Problem/Trouble reports, e.g., library model changes and product bugs
- New 'wrapper' software specifications

6 Threaded Processes

6.1 Overview

Threaded Processes define the procedures associated with those activities that span the major library development processes. Included in this chapter are definitions for:

- Software Development Folder (SDF)
- Configuration Management (CM)
- Configuration Management Request (CMR) Form
- Component Status Tracking
- Library Development Quality Control

6.2 SDF

6.2.1 Purpose

The SDF is an organizational tool used to provide a repository for all of the library data (e.g., documents and files) including the library model(s) and software components (products).

6.2.2 Policy

- All data in the library will be organized in SDFs.
- Each file will be uniquely identifiable by its SDF ID.
- The SDF and their contents are under library development CM control.

6.2.3 Procedures

6.2.3.1 SDF Organization and Contents

A component SDF contains the following files:

- Component Status Sheet
- Component Data Sheet

- Common Criteria Evaluation
- Domain Criteria Evaluation
- Component Evaluation Report
- References to applicable problem reports (STRs) (See Appendix G) and system change requests (SCRs)(See Appendix F)
- Product documentation [or references to where the documentation can be found] (e.g., requirements specifications, user's guides)
- CMRs
- Product code [or references to how the code can be obtained] (i.e., source, object, and executable)

6.2.3.2 SDF Creation

An SDF is created for every component when its first data/file is created. For components, SDFs are created during the acquisition phase of the library population process (See Section 5.2.1). When an SDF is created, it is assigned a unique identifier (See Appendix B) that can be used to explicitly reference its contents.

The SDF is under constant library development CM control. The level of control depends on the maturity of the item represented in the SDF and is defined in the CM (See 6.3) section of this chapter.

6.2.3.3 SDF Updates

As each new file is created for the library model or component, it is included in the SDF and placed under CM control. If a file needs to be updated, it is done in accordance with the established CM controls so that all revisions of each file remain in the SDF.

6.3 CM

6.3.1 Purpose

The purpose of CM is to provide version control of any files in the baselined developmental released library. Configuration management control is placed on the files associated with the library model and component files.

6.3.1.1 Definitions

The library model and the component files could be part of either a developmental baseline or a release baseline, as defined below:

Developmental Baseline. Once a file is ready for distribution/use among developers, it is part of the developmental baseline. The most recent versions of each file is part of the developmental baseline.

Release Baseline. Once the LCB and CCB decide to release a version of the developmental base line to the operational library, it becomes the release baseline. (Refer to Configuration Management section of Volume I of LOPP)

6.3.2 Policy

Each version of a controlled file will be re-creatable.

The versions of each file in a baseline (development and release) shall be known.

All activities affecting the contents of either the developmental baseline or the release baseline must be accompanied by a CMR Form (See Appendix H).

Any activity affecting the release baseline requires LCB approval.

Any cost or schedule-bearing activity requires pre-approval by the LCB.

6.3.3 Procedures

The developmental baseline is established while the item is in the development stages and the release baseline is established (by the LCB and CCB) when the product is to be released to users. The point at which the developmental baseline is to be established is at the developer's discretion. The two fundamental procedures involved in CM are check-in and check-out. To place a file under CM, it must be checked-in to the CM system being used.. Once a file is checked-in, a check-out must be performed to access/augment it. CMR approval is required prior to the check-out of any file under CM. At the time of check out, a time limit is established. Once modifications are complete, the file is returned to the CM system via the check-in procedure.

6.4 Configuration Management Request (CMR) Form

6.4.1 Purpose

The purpose of the CMR Form is to track check-in and check-out requests to baselined items, e.g., the domain model, developmental software and library software. The CMR may be initiated due to either change requests or problem reports on released items.

6.4.2 Policy

All changes/enhancements to baselined items will be documented in a SCR (See Appendix F). All perceived problems to baselined items will be documented in a STR (See Appendix G). All constituent files of the developmental library will be controlled and tracked using a CMR Form (See Appendix H).

6.4.3 Proce

6.4.3.1 CMR Contents

The CMR contains information such as

- Name of originator/reviewer.
- Type of request (check-in, check-out, etc.).
- Reason for check-in or check-out.
- Status (being evaluated, on hold, accepted, rejected, completed, etc.).

6.4.3.2 CMR Process

The following is a description of the CMR process, as depicted in Figure 6-1:

1. Library users or library personnel submit an STR/SCR to the operational library CM.
2. The operational library CM reviews and forwards the appropriate STR/SCR to the library development CM.
3. The SCR or the STR is assigned to an analyst. After analyzing the SCR or STR, the analyst determines the affected components or portions of the library.
4. The developer generates the CMR by filling in the title and description of the reason for requesting a component and submits the CMR to library development CM. Associated SCRs and/or STRs are noted in the CMR.
5. If the CMR is rejected, the analyst is notified along with an explanation. The CMR is closed.
6. If the CMR is accepted, the requested items are checked-out to a library developer.
7. The developer completes the requested work and submits the updated CMR and the items to the library development CM.
8. Library development CM reviews the updates and checks-in the items. The CMR is closed.

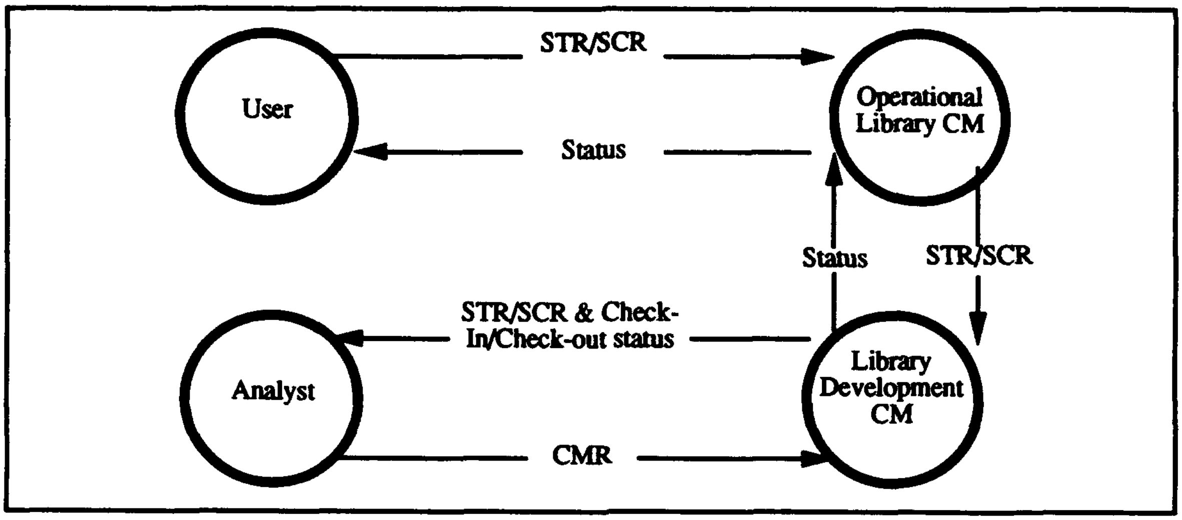


Figure 6-1 Configuration Management Request Process.

6.5 Component Status Tracking

6.5.1 Purpose

There are many components present in the library. The current status of each component being developed should be readily determinable to facilitate the management of library development.

6.5.2 Policy

The status of each component in the library development process is tracked on a Component Status Sheet.

6.5.3 Procedures

As each development step is completed, the completion date and developer are noted in the indicated place on the Component Status Sheet.

1. Component Data Sheet completed.

2. Software Development Folder created.

3. Domain criteria evaluation completed.

4. Common criteria evaluation completed.

5. Component Evaluation Report completed.

6. LCB's component acquisition decision reached.

7. Adaptation design completed and reviewed.

8. Adaptation code completed and reviewed.

9. Adaptation stand-alone testing completed.

10. Integration testing completed.

11. LCB's final acceptance/rejection of component.

Additionally, the identification and status of all change requests and trouble reports associated with the component should be kept up-to-date on the Component Status Sheet.

6.6 Library Development Quality Control

6.6.1 Purpose

The purpose of library development quality control is to monitor library development activities to continually evaluate and improve the library development process.

6.6.2 Policy

The library development organization will have a semi-autonomous quality assurance group. The group participates with all members of the library development organization to ensure established policies and procedures are executed properly. The quality assurance group also identifies and documents quality issues and recommends changes to the LCB.

6.6.3 Procedures

There are two facets to the quality control procedure: product quality and process quality. A quality process ultimately produces quality products. The goal of the quality control is to monitor the process and suggest refinements that will positively affect the quality of the products. Thus, the semi-autonomous quality group acts as a feedback loop in every library development process. Some types of data on which the quality group will base their recommendations include:

1. Trends in error sources identified by change requests and trouble reports.

2. Turnaround times on changes.

3. Analysis of lessons learned data.

APPENDIX A - Component Status Sheet

A.1 Purpose

The purpose of the Component Status Sheet (CSS) is to have a current record for each component in the development library. This Form is created for each component during the acquisition process and updated throughout the development of the component. This Form enables the developers to readily provide status reports on all components under development.

A.2 Instructions

The CSS is created for a component once it passes the screening phase of acquisition and is recommended for detailed evaluation. As each step in the component development is completed, the developer notes their name and completion date on the status sheet. The CSS is maintained in the component's Software Development Folder (SDF).

The following is a step-by-step guide to filling out the CSS. Each entry on the form must contain the name/initials of the person who did the work or the person who is making the entry.

1. Upon creation, enter the component name, SDF identifier, Component Data Sheet date.

2. Enter the date the SDF was created.

3. After domain criteria evaluation is performed, enter the domain criteria completion date.

4. After the common criteria evaluation is performed, enter the common criteria completion date.

5. Enter the evaluation report date when completed.

6. Enter the date the component was presented to LCB for consideration.

7. Enter the LCB decision and date.

8. If adaptation is required, enter the date when the adaptation specification is completed. If this is a new component, then identify by component name and SDF identifier.

9. When adaptation design is reviewed and completed, enter date.

10. When adaptation coding is reviewed and completed, enter date.

11. When adaptation stand-alone testing is completed, enter date.
12. When component integration is completed, enter date.
13. When integrated component is presented to LCB for final consideration, enter date.
14. Enter LCB decision to add to the library and date.

Component Status Sheet

Component Name:

SDF #:

Phase	Activity Completed	Name	Date	Comments
Acquisition	Component Data Sheet			
Acquisition	Software Development Folder created			
Acquisition	Domain Criteria completed			
Acquisition	Common Criteria completed			
Acquisition	Evaluation Report completed			
Acquisition	Component proposed to LCB			
Acquisition	LCB Acquisition Decision			
Acquisition	Adaptation Specification completed			
Adaptation	Design reviewed and completed			
Adaptation	Coding reviewed and completed			
Adaptation	Stand-Alone Testing completed			
Integration	Integration completed			
Integration	Integration Report sent to LCB			
Integration	Component added to library			

Current Changes/Problems:

Fig. A-1

APPENDIX B - Component Data Sheet

B.1 Purpose

The purpose of the Component Data Sheet is to obtain information from vendors and sources to initiate the component evaluation process. This is the first step in the acquisition of an 'off-the-shelf' software product. After product identification, the Component Data Sheet is used to generate a prioritized list of components, so the evaluations can be conducted with a high acceptance rate, since components deemed not suitable to the domain will be given a low priority.

B.1.1 Instructions

1. Retrieve a blank Component Data Sheet.
2. Assign a unique SDF identifier (See Section 1.2).
3. Contact the vendor of the product to request all available marketing literature.
4. Fill in the name of the product.
5. Fill in the version number.
6. Fill in the Component Class the product fits under.
7. Fill in the name of the product's developer.
8. List any required support software.
9. List any available enhancement software.
10. Fill in any information regarding documentation.
11. List any available information regarding programmer's interfaces or libraries included with the software.
12. List the requested demonstration information, if necessary.
13. Fill in the chart regarding the operating systems and hardware supported by the application.
14. Fill in the chart regarding the windowing environments supported by the application.

15. Detail the license information. Include an attachment if necessary.

16. Fill in the chart regarding the points of contact for CARDS members, government, and commercial sites. If applicable, include an e-mail address to contact for information/support.

17. As necessary, list the contact dates and information regarding the contact.

18. Include any product notes deemed appropriate.

B.1.2 SDF Identification

Each SDF will have a unique identification of the following format:

- 3 letter domain ID (e.g., GCC)
- a dash
- 3 letter concept name (e.g., DTP, or BRS)
- a dash
- one letter software type code (C= COTS, G= GOTS)
- unique 5 digit number for each SDF in the particular Concept
- a dash
- 6 digit date in the form MMDDYY the SDF was created

An example SDF ID for a commercial desktop publishing tool being acquired for the CARDS Command Center domain would be GCC-DTP-C-00045-010293.

Component Data Sheet

SDF#:

Product Name:
Version #:
Component Class:
Developer:
Product Category (e.g., COTS, GOTS, Public Domain):

Support Software Required:

Available Enhancement Software:

Documentation Available:

Programmer's Interfaces/Libraries:

Demo Information (availability, cost, content, utility, media, etc.):

Operating Systems/Hardware Supported

Operating System	OS Version	Hardware Platform	Demo? (Y/N)

Window Environments Supported

Window System	Version

Licensing Information:

GSA Contract Available (Y/N)?

Fig. B-1

Component Data Sheet

Points of Contact

Affiliation	Contact Person	Phone	Address
CARDS			
Vendor (for goverment)			
Vendor (for goverment)			
Vendor (for commercial)			
Vendor (for commercial)			
FTP SITE			
FTP SITE			

Contact Log

Date	Contact	Affiliation	Reason

Notes

Fig. B-2

APPENDIX C - Domain Criteria Evaluation

C.1 Purpose

The purpose of a domain criteria evaluation is to evaluate prospective software products against domain-specific criteria. The result of the evaluation is used to make an objective recommendation concerning the inclusion of the software product into the library.

C.1.1 Instructions

1. Retrieve a blank copy of the domain criteria form (see pages C-2 and C-3) to be used in the evaluation. A library data model will contain many forms, each one specifically designed and associated with an individual generic concept defined.

2. Install and execute a copy or demonstration of the product, if possible. Run the program through some typical exercises to get a feel for the functionality provided.

3. Answer the questions based on the program itself (if available), the documentation provided or the demonstration. For those questions which the evaluator has comments, use the space provided on the right side of the document or attach an explanation at the end.

4. Contact the vendor or the developer for those items which were not answered by the documentation or the demonstration.

5. For those questions which are considered "Critical," any answer of "No" has to be resolved in either the "Comments" section or on an attachment. The library may contain components that do not fully meet the domain criteria.

6. After the questions are answered, the evaluator records the component domain criteria information in the appropriate file and initiates the Component Evaluation Report for this component. The evaluator records:

   A. a summary of any problems encountered during the component domain criteria evaluation.

   B. a suggestion of possible solutions to any of the problems. Note: any item marked "Critical" which has an answer of "No" and no feasible resolution may be rejected in the final analysis.

7. Update the Component Status Sheet for this component.

C.1.2 Domain Criteria Form

The following pages contain a template of a domain criteria form.

Component Domain Criteria

SDF#:XXX-YYY-Z-00000-MMDDYY

Ref #	(insert Class here)Domain Criterion	Response	Critical To Command Center	Criterion Source	Response Source	Comment
Component Constraints						
1						
Architectural Constraints						
2						
Implementation Constraints						
3						

Criteria Sources

Response Sources

References

CCDH	Command Center Design Handbook. AVS/PRISM RA/AD Team. Revision date: September 1991.
GCCAR	Lonardo, G.G. and Wallin, J.D. Generic Command Center Architecture Report for the Portable, Reusable, Intergrated Software Modules (PRISM) Program. Prepared for : Electronic Systems Center (ESC) Air Force Materials Command, USAF, Hanscom AFB, MA 01731-5000. Revision date: 11 August 1992.

Comments

Fig. C-2

APPENDIX D - Common Criteria Evaluation

D.1 Purpose

The purpose of the common criteria evaluation is to assess the prospective software component for reliability, maintainability, usability, maturity, portability, cost, security and performance. The result of the evaluation is used to make an objective recommendation concerning the inclusion of the software component into the library. The common criteria that a component meets are utilized by the users in choosing appropriate components from the library.

D.1.1 Instructions

1. Retrieve a blank copy of the component common criteria form.
2. Fill out the component information.
3. Fill out the required information. Much of this data can be obtained from the Component Data Sheet, vendor marketing literature, or by contacting the vendor/developer.
4. After the questions are answered, the evaluator saves the component common criteria information in the appropriate file and then completes the Component Evaluation Report.
5. Update the Component Status Sheet for this component.

D.1.2 Example Common Criteria Form

The following pages contain an example common criteria form. This form is designed for evaluation of software for the CARDS Command Center Library.

Component Common Criteria

SDF #:

Component Name:

Ref #	Common Criterion	Source/ Method[1]	Data/Value
	User Support		
1	Does the vendor offer User/Customer Support?		
2	Does the product offer on-line help?		
3	Is there User Documentation (electronic or hardcopy)?		
4	Are there Programmer's Interfaces or Libraries available?		
5	Is training offered by vendor?		
6	List the costs associated with the various license and maintenance agreements.		
7	Are GSA Contracts available?		
	Platform		
8	On what platforms does the program run (VAX, PCs, etc.)?		
9	List the standards that the product supports in the following categories:		
9.1	Network Capabilities (e.g., Novell, Banyan)		
9.2	Operating System Compliance (e.g., POSIX)		
9.3	Database Query Languages (e.g., SQL)		
9.4	Windowing Systems (e.g., Motif, Open Look)		
10	Has the product been ported from another operating system/platform?		
	Miscellaneous		
11	Is there any additional software needed in conjuction with the product?		
12	Does the product satisfy MLS Standards?[2]		

Fig. D-1

Component Common Criteria

SDF #: Component Name:

Ref #	Common Criterion	Source/ Method[1]	Data/Value
13	Is the component a "trusted product" with respect to security (i.e., evaluated by the National Computer Security Center)?		
14	How long has the product been in release?		
15	Is the product an initial release (if not, explain)?		
16	Number of licenses granted.		Date confirmed:
17	If source code is supplied, list the language of implementation and answer the next three questions.		
18	If source code is supplied, is adequate commenting provided in the code?[3]		
19	Is there a list of known errors for the current version of the product? If yes, attach explanations.		
20	Compile results from Benchmark reports on the product, Product Literature, Technical & Trade publications on product performance.		
	Source Code		
21	Has the product been field-tested?		
22	Support hardware required.		
23	Support software required.		
24	Please provide a brief description of the wrapper/source code functions in an attachment.		
25	Have the machine/environment dependencies been modularized?		
26	Does the wrapper/code have test suites?		
26.1	Have the test suites been run on the wrapper/code?		
26.2	Has the wrapper/code successfully passed through the test suites?		

Fig. D-2

Component Common Criteria

SDF #:

Component Name:

Ref #	Common Criterion	Source/ Method[1]	Data/Value
27	Do performance tests exist for the wrapper/code?		
27.1	Has the wrapper/code successfully passed the performance tests?		

NOTES

1) The codes contained in the Source/Methods column above are interpreted as follows:

- * VML: Vendor Marketing Literature - Brochures and other literature supplied by the vendor as part of their standard marketing approach.
- * V: Vendor - Additional reports and information on the product supplied by the vendor on request.
- * DEMO: Running demo version of the product
- * FULL: Running fully functional version of the product
- * USER: Compiling reports on the product by established user base and verifying the reports
- * WORK: Analyzing product's implementation so as to be able to recommend acceptable workarounds
- * PR: Analyzing product reports in Trade Magazines
- * DOCS: Product documentation
- * CODE: Source code/in-line comments
- * TEST: On-site performance testing

2) The Multi-Layered Security (MLS) standards establish requirements to be applied during the acquisition, development, or support of software systems. The requirements of this standard apply to the development of Computer Software Configuration Items (CSCIs). This standard applies to the extent specified in the contract clauses, the Statement of Work (SOW), and the Contract Data Requirements List (CDRL). For more information, see the military document DOD-STD-2167A.

3) This criteria is currently in the evaluator's opinion, with rationale provided below:

Fig. D-3

APPENDIX E - Component Evaluation Report

E.1 Purpose

The purpose of the Component Evaluation Report is to provide a standardized format for reporting the step-by-step conclusions drawn from the various responses to the domain and common criteria questions. The Component Evaluation Report is also used in the process of recommending the individual components for inclusion into the library to the Library Control Board (LCB).

E.1.1 Instructions

1. Upon completion of the domain criteria evaluation, create the Component Evaluation Report by obtaining a blank copy (see pages E-2 - E-4) and placing it into the component's SDF.

2. Fill in the evaluation report by following the instructions contained within the report template. The domain criteria section of the Form should be annotated with the findings of the domain criteria evaluation including a discussion of any major problems identified.

3. Upon completion of the common criteria evaluation, complete the Component Evaluation Report by filling in the common criteria section and the recommendation section.

4. Complete the Executive Summary section. The Executive Summary should include a brief overview of both the domain and common criteria sections and the recommendation made to the LCB regarding the inclusion of the product in the library.

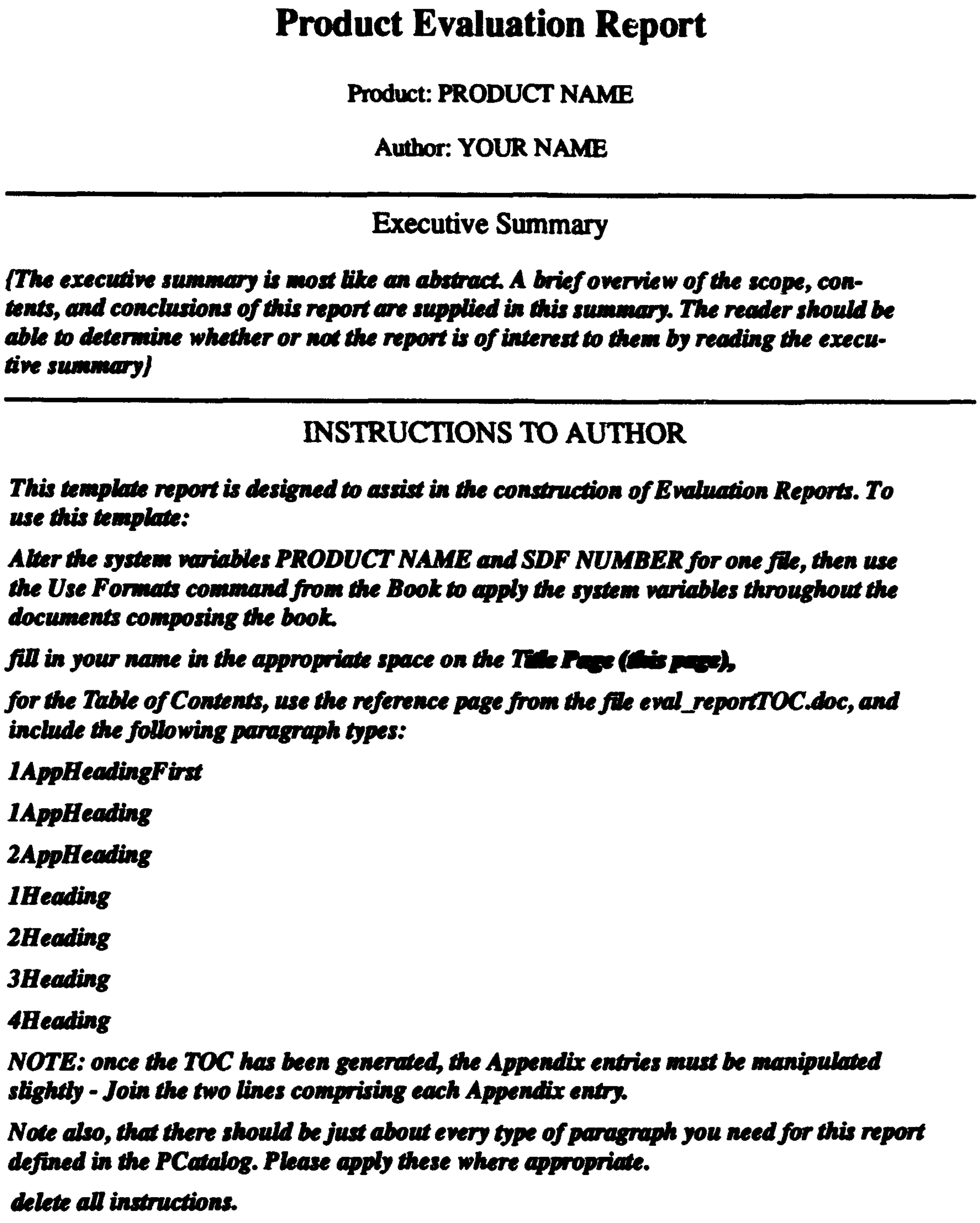
5. Delete all of the instructions from the report.

6. Compile the book containing all the files associated with the report and the completed forms.

7. File the completed Component Evaluation Report in the SDF, submit the report to the LCB for consideration as a new library component, and update the Component Status Sheet.

E.1.2 Report Template

The following is the template for the Component Evaluation Report used in the CARDS Command Center Library.



Product Evaluation Report

Product: PRODUCT NAME

Author: YOUR NAME

Executive Summary

{The executive summary is most like an abstract. A brief overview of the scope, contents, and conclusions of this report are supplied in this summary. The reader should be able to determine whether or not the report is of interest to them by reading the executive summary}

INSTRUCTIONS TO AUTHOR

This template report is designed to assist in the construction of Evaluation Reports. To use this template:

Alter the system variables PRODUCT NAME and SDF NUMBER for one file, then use the Use Formats command from the Book to apply the system variables throughout the documents composing the book.

fill in your name in the appropriate space on the Title Page (this page),

for the Table of Contents, use the reference page from the file eval_reportTOC.doc, and include the following paragraph types:

1AppHeadingFirst

1AppHeading

2AppHeading

1Heading

2Heading

3Heading

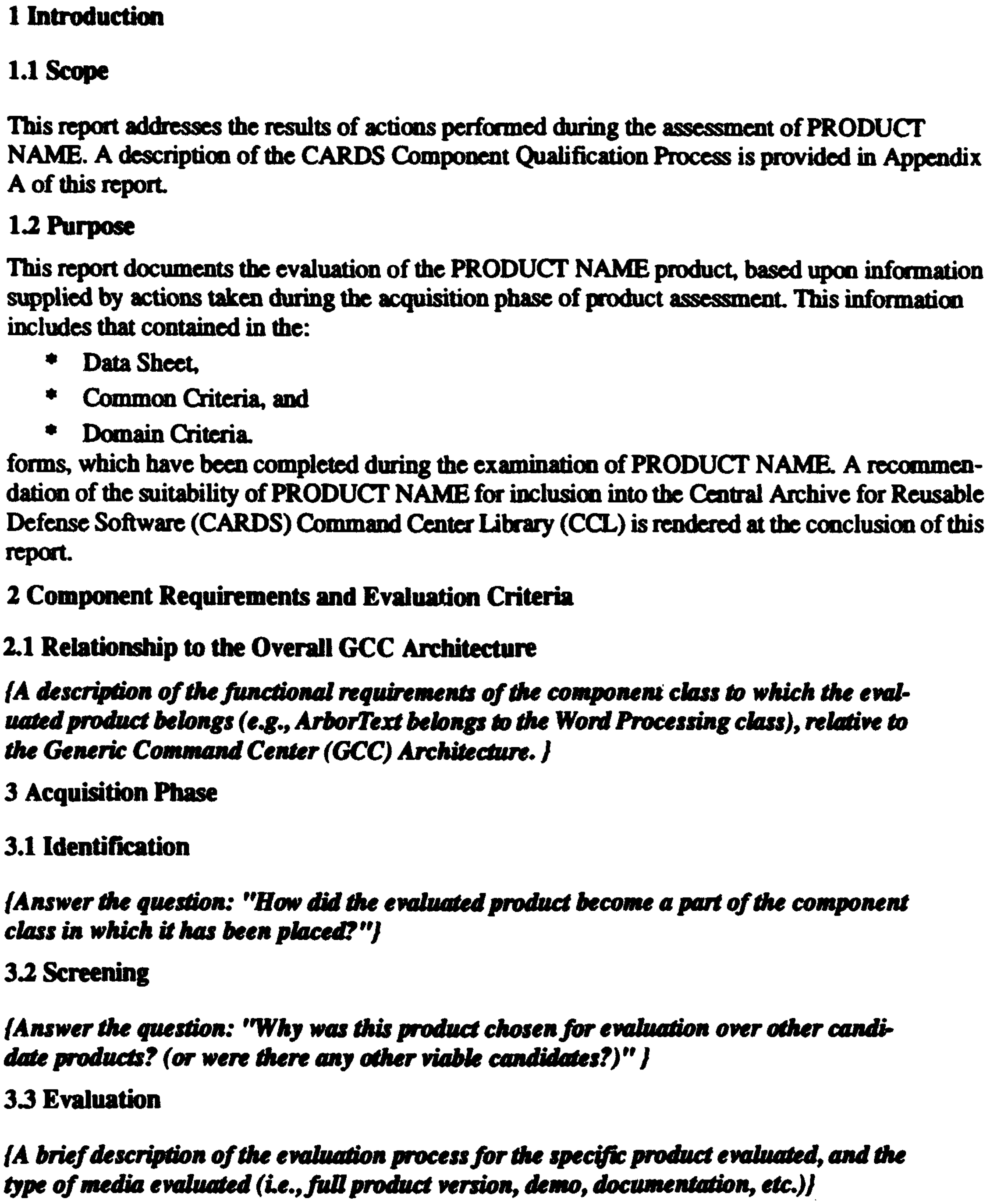
4Heading

NOTE: once the TOC has been generated, the Appendix entries must be manipulated slightly - Join the two lines comprising each Appendix entry.

Note also, that there should be just about every type of paragraph you need for this report defined in the PCatalog. Please apply these where appropriate.

delete all instructions.

Fig. E-1



1 Introduction

1.1 Scope

This report addresses the results of actions performed during the assessment of PRODUCT NAME. A description of the CARDS Component Qualification Process is provided in Appendix A of this report.

1.2 Purpose

This report documents the evaluation of the PRODUCT NAME product, based upon information supplied by actions taken during the acquisition phase of product assessment. This information includes that contained in the:

- Data Sheet,
- Common Criteria, and
- Domain Criteria.

forms, which have been completed during the examination of PRODUCT NAME. A recommendation of the suitability of PRODUCT NAME for inclusion into the Central Archive for Reusable Defense Software (CARDS) Command Center Library (CCL) is rendered at the conclusion of this report.

2 Component Requirements and Evaluation Criteria

2.1 Relationship to the Overall GCC Architecture

{A description of the functional requirements of the component class to which the evaluated product belongs (e.g., ArborText belongs to the Word Processing class), relative to the Generic Command Center (GCC) Architecture. }

3 Acquisition Phase

3.1 Identification

{Answer the question: "How did the evaluated product become a part of the component class in which it has been placed?"}

3.2 Screening

{Answer the question: "Why was this product chosen for evaluation over other candidate products? (or were there any other viable candidates?)" }

3.3 Evaluation

{A brief description of the evaluation process for the specific product evaluated, and the type of media evaluated (i.e., full product version, demo, documentation, etc.)}

Fig. E-2

3.3.1 Domain Criteria Evaluation

{A description of the fulfillment of the domain criteria by the evaluated product. Use subheadings (paragraph format 4Heading in FrameMaker) germane to the specific domain criteria being reviewed, as each class has its own particular areas of interest.}

3.3.2 Common Criteria Evaluation

The following is a description of the fulfillment of the common criteria for a reusable software component by PRODUCT NAME. The completed Common Criteria Form is contained in Appendix C of this report.

{A description of the fulfillment of the common criteria by the evaluated product.}

3.3.2.1 Reliability

3.3.2.2 Maintainability

3.3.2.3 Usability

3.3.2.4 Maturity

3.3.2.5 Portability

3.3.2.6 Cost

{Description of the licensing information}

3.3.2.7 Security

4 Adaptation

{State the recommended adaptations, or state that no adaptations are necessary. DELETE the following statement if no adaptations necessary.}

At the time of this report, none of the above adaptations to PRODUCT NAME have been implemented.

5 Integration

At the time of this report, the integration of PRODUCT NAME into the CARDS CCL has not occurred.

6 Recommendation

{The recommendation for inclusion of this product into the CARDS CCL. Note that this recommendation should be included (in brief) in the Executive Summary}

Fig. E-3

APPENDIX F - Software Change Request (SCR) Form

F.1 Purpose

The SCR is used by the library users and library personnel to request changes to operational released versions of library software. The SCR identifies the changes and its perceived impacts on the system. The SCR also gives an estimation of resources required to implement the changes.

F.1.1 Instructions

The SCR is used to request changes to the established developmental configuration software programs. These changes could be in the form of additions, deletions or modifications to an existing capability. The SCR will also be implemented to maintain changes of comp?:bility with other system interfaces.

The instructions for SCR preparation are as follows:

1. SCR NO - A unique number assigned to a SCR by the configuration manager. The originator of the request leaves this field blank.

2 SOURCE - Originator enters CARDS or User

3. HOTLINE REPORT NUMBER - If the SCR originated via a hotline report, the hotline report number is entered here by the hotline operator. Otherwise, the field is not used.

4. CHANGE NAME - Originator enters a brief phrase description of the change.

5. CHANGE DESCRIPTION - Originator enters a description of the change, including all items to be changed, added, or deleted. The relationship to other changes, reported problems and modifications should be included.

6. ORIGINATOR - Originator enters their name.

7. ORIGINATOR DATE - Originator enters the date the report is submitted.

Once these fields are completed, this form is mailed to cm@cards.com. The sender of the SCR will be notified of its receipt and the assigned SCR number. Modifications to the library software are made via official releases. The release notes accompanying each release indicate which SCRs were incorporated in the release. The remainder of the form is to be completed by the library development staff.

8. SCR STATUS - Analyst/CM/Software lead enters SCR status as follows:

- DUP = duplicate of another SCR (cite number).
- DONE = completed and documented, QA approved but not released.
- GSCCB = under evaluation for proper disposition by the authorizing governmental official.
- HOLD = to be considered at a later date for implementation.
- REL = implementation included in current release.
- RET = not to be implemented; retired.
- CCCB = under evaluation by the CARDS Configuration Control Board for proper disposition.
- LCB = under evaluation by the Library Control Board for proper disposition.
- TBIMP = approved and placed in the implementation cycle.
- UDINV = under investigation.
- INTST = in test.
- RLF = forwarded to STARS.

9. ANALYST ASSIGNED - Software lead enters the name of the individual assigned to analyze the request.

10. ANALYSIS - Analyst enters analysis of the proposed change, the rec~~mmended solution and alternative solutions, if available.

11. RELATED STR OR SCR - Analyst enters related SCR or STR, if any.

12. DEVELOPMENT REQUIREMENTS - Analyst identifies resource requirements for the proposed change (including manpower estimates). The resource requirements should include those needed for the redesign, recoding, testing, installation, and adaptation required to implement the change.

13. ALTERNATIVES/IMPACTS - Analyst/software lead enters the cost, schedule, and interface impacts if the solution is approved. Performance impacts if the solution is not approved must also be entered. As applicable, the impact on other configura'ion items, other work, system employment, system resources, training, etc., are also entered.

14. RECOMMENDED IMPLEMENTATION POINT - Analyst include a recommended implementation point. A specific release number or date may be included. The rationale for the implementation of the change before, with or after some other system modification.

15. BASELINE AFFECTED - Analyst enters the software release and revision number of the system that will be affected by the change.

16. CONFIGURATION ITEMS IMPACTED - Assigned analyst enters software release and revision number affected by the change as well as the specific configuration items affected by the change. This includes documentation, installation procedures, etc., as well as software.

17. ACTION TAKEN - Analyst/implementor describes what was actually done to implement the change.

18. TEST PLAN - Analyst/implementor and quality assurance describe the test plan used to verify the satisfactory resolution of the SCR.

19. TEST RESULTS - Analyst/implementor and quality assurance describe the results of the test used to verify the satisfactory resolution of the SCR.

20. ANALYSIS APPROVAL - Name of the software lead who approved the analysis and recommended solution for the change. Also fill in the DATE of the approval in the field below.

21. COMPLETION APPROVAL - Name of the software lead who approved that the SCR was satisfactorily resolved. Also fill in the DATE of the approval in the field below.

22. QA APPROVAL - Name of the tester/QA lead who approved that the SCR was satisfactorily resolved. Also fill in the DATE of the approval in the field below.

23. DATE CLOSED - Enter the date the SCR is closed.

F.1.2 Software Change Request Template

See next page for an SCR form.

APPENDIX G - System Trouble Report (STR) Form

G.1 Purpose

The STR can be initiated when a perceived problem in the library is encountered by library users or library personnel. The intent of the STR is to log and track all problems in the library to resolution.

G.1.1 Instructions

The instructions for the STR preparation are as follows:

1. STR NO - Is a unique number assigned to a System Trouble Report by the configuration manager. The originator of the request leaves this field blank.

2. SOURCE - Originator enters CARDS or User.

3. HOTLINE REPORT NUMBER - If the STR originated via a hotline report, the hotline report number is entered here by the hotline operator. Otherwise, the field is not used.

4. PROBLEM NAME - Originator enters a brief phrase describing the problem.

5. PROBLEM DESCRIPTION - Originator enters a description of the problem and the conditions, inputs, and equipment configuration when the problem arose. A description of the activities leading up to the problem occurrence with sufficient problem information to permit duplication and analysis should be reported. The relationship to other reported problems and modifications should be included if known. Any error messages displayed when the error occurred should be reported.

6. SYSTEM STATUS - Originator enters system status at time of problem occurrence, i.e., Is the system running optimally.

7. ORIGINATOR - Originator enters their name.

8. ORIGINATION DATE - Originator enters the date the report is submitted.

9. CATEGORY - Originator enters the defect category assigned to the STR, if known. Otherwise, the field is completed by the analyst assigned to work on the STR. The software lead reviews the determination and revises it if necessary. Category is indicated by a one-character field as defined below:

   - S = Software Defect. The software does not operate according to the documentation, and the documentation is correct.

- D = Documentation Defect. The software does not operate according to the supporting software documentation, but the software operation is correct.
- E = Design Defect. The software operates according to the supporting software documentation, but a design error exists.
- M = Library Model Defect
- R = RLF Defect

Once these fields are completed, this form is mailed to cm@cards.com. The sender of the STR will be notified of its receipt and the assigned STR number. Corrections to the library software are made via official releases. The release notes accompanying each release indicate which STRs were fixed in the release.

10. STR STATUS - analyst/cm/software lead enters STR status as follows:

- DUP = duplicate of another STR (cite number)
- FIXED = fixed and documented, QA approved but not released
- GSCCB = under evaluation for proper disposition by the authorizing governmental official.
- HOLD = to be considered at a later date for implementation
- REL = fix or implementation included in current release
- RET = not to be implemented; retired
- CCCB = under evaluation by the CARDS Configuration Control Board for proper disposition
- LCB = under evaluation by the Library Control Board for proper disposition
- TBIMP = approved and placed in the implementation cycle
- UADUP = unable to duplicate the problem
- UDINV = under investigation
- RLF = forwarded to STARS
- INTST = in test

11. PRIORITY - Analyst or CM enters priority assigned to the STR. This priority is subject to review by LCB. Priority is indicated by a one digit number as defined below:

    - 1 = Prevention of Essential Function. The prevention of the accomplishment of an essential function in accordance with requirements; interferes with the user to the extent that the user cannot accomplish an essential function.

    - 2 = Essential Function Degraded with no work around. The performance of an essential function is less than required and no alternative exists.

    - 3 = Essential Function Degraded with work-around. The performance of an essential function is less than required but a work-around exists.

    - 4 = Inconvenience. An error that causes a problem but does not affect essential functions.

    - 5 = Other. Any other error not applicable to the those above (e.g., documentation).

12. ANALYST ASSIGNED - Software lead enters the name of the individual assigned to analyze the problem.

13. PROBLEM DUPLICATION - Analyst indicates if the problem can be duplicated.

14. ANALYSIS - Analyst enters analysis of the problem, the recommended solution and alternative solutions, if available. The nature of the recommended solution should be described when applicable, to support the rationale and test results.

15. CONFIGURATION ITEMS IMPACTED - Analyst and/or problem corrector enters items impacted by the proposed analysis solution to the STR.

16. CORRECTIVE ACTION - Analyst and/or problem corrector(s) enter corrective actions taken in the resolution of the problems.

17. TEST PLAN - Analyst, problem corrector(s), and quality assurance describe the test plan used to verify the satisfactory resolution of the STR problem.

18. TEST RESULTS - Analyst, problem corrector(s), and quality assurance describe the results of the test used to verify the satisfactory resolution of the STR problem.

19. ANALYSIS APPROVAL - Name of the software lead who approved the analysis and recommended solution to the problem. Also fill in the DATE of the approval in the field below.

20. COMPLETION APPROVAL - Name of the software lead who approved that the STR was satisfactorily resolved. Also fill in the DATE of the approval in the field below.

21. QA APPROVAL - Name of the tester/QA lead who approved that the STR was satisfactorily resolved. Also fill in the DATE of the approval in the field below.

G.1.2 System Trouble Report (STR) Template

See next page for an STR form.

SYSTEM TROUBLE REPORT

STR NO:

SOURCE:
HOTLINE REPORT NUMBER:

PROBLEM NAME:

PROBLEM DESCRIPTION:

SYSTEM STATUS:

ORIGINATOR:
ORIGINATION DATE:

CATEGORY:

= REMAINDER OF FORM TO BE COMPLETED BY LIBRARY DEVELOPMENT STAFF =

STR STATUS:
PRIORITY:

ANALYST ASSIGNED:

PROBLEM DUPLICATION:

ANALYSIS:

CONFIGURATION ITEMS IMPACTED:

CORRECTIVE ACTION:

TEST PLAN:

TEST RESULTS:

Fig. G-1

APPROVALS

ANALYSIS APPROVAL:
DATE:

COMPLETION APPROVAL:
DATE:

QA APPROVAL:
DATE:

Fig. G-2

APPENDIX H - Configuration Management Request (CMR) Form

H.1 Purpose

The CMR form aids in the CM control of the developmental library. By helping to track and controll developmental library files, this Form accompanies every request for library model check-in and check-out.

H.1.1 Instructions

An annotated version of the CMR is shown below to indicate how the form is to be filled out:

Originator: Your name

Type and time: Either INIT or CO or CI as explained below and estimate of the time files will be checked out if type is CO.

Valid types are INIT (initial check in), CO (check out for modification), and CI (check in of modified files). For CO, indicate an estimate of the time the files will be checked out.

SCR/STR Reference and Location:

All work done for the library in response to either an STR or an SCR. At the point that CO is requested, the analysis portion of the SCR/STR should be completed to indicate what work is to be done.

At the point that CI is requested, the SCR/STR should be updated to reflect the work that was done and to include a test plan.

In this field, indicate where (under AFS) your copy of the STR/SCR, with the analysis and test plan sections completed, is/are located.

Files (include full path name) and other pertinent information:

- For CO requests, list all files to be checked out - give full path names. For CI requests, give current location of the file, the full pathname for where the file is to be checked in, and the log message/initial message to be used when files are checked into RCS.

- For INIT requests, give current location of the file and the full pathname for where it should be entered into the src area. Also, indicate where the file should go in the built library.

- Also include any other information or comments that you wish in this area.

APPENDIX I - CARDS Software Documentation Standard

I.1 Purpose and Scope

This standard defines the documentation requirements for CARDS library developed software. The use of this documentation standard will:

1. Ensure completeness of documentation
2. Improve software maintainability
3. Promote quality software development practices

This standard only addresses the documentation requirements for software developed for the operational library. It does not address software design methods, coding standards, or software development plans.

I.1.1 Applicability

This standard applies to all software developed for the operational library. As applicable, this standard may also be used as evaluation criteria for externally developed software, for which source code and other life-cycle documentation is available.

Software includes both code and data files. Among the items that are considered software:

1. Library components
2. Library model including:
   - SNDL scripts
   - RBDL scripts
3. Library infrastructure software including:
   - Application installation and configuration files
   - Library tools, e.g., interoperability, system composition

I.1.2 Responsibility

The responsibility for following this standard lies in the hands of the software developers and their management. Within the CARDS library, software can be developed within a number of functional areas such as system administration, configuration management, and library development.

I.1.3 Policy

All software developed for the CARDS Program shall be documented in a consistent manner that enables in-process evaluation and post-development maintenance.

I.1.4 Background

This standard addresses the following five phases of the software life-cycle:

- Requirements Analysis
- Design
- Code and Debug
- Integration
- Maintenance

Each phase produces certain products whose content and format should be standardized. Figure I-1 identifies the documents to be produced during each phase of the software development.

The documents Figure I - 1 represent, to a large degree, the documents defined by DOD-STD - 2167A. As each document is described in subsequent sections, the applicable 2167A Data Item Description (DID) will be identified. The DID can be referenced to aid in producing the subject document.

For convenience, the documentation is described in terms of documents. For large software items, separately bound documents may be appropriate. For small products (i.e., software utilities), it is reasonable to combine many of the documents within a single file. A suggested minimum number of documents (files) would be three (3):

1. Requirements, design, user, and programmer data supplied in the header of the source code file(s).
2. Version Description Document
3. Descriptions and reports for stand-alone and integration tests

I.1.5 Standards

I.1.5.1 Requirements Analysis Phase

During the requirements analysis phase, the subject software program's requirements are determined. Two documents are produced during this phase:

1. **Software Requirements Specification** - This document contains the performance and functional requirements of the software. All design and implementation constraints should also be noted along with requirements for portability and adaptability. [2167A DID: DI-MCCR-80025]

2. **Interface Requirements Specification** - This document defines the requirements of interfaces to other software programs, to hardware, and to the user. [2167A DID: DI-MCCR-80026].

I.1.5.2 Design Phase

During this phase, the implementation structure of the software product is formulated. Depending on the size of the product, this may occur in two sub-phases: top level design and detailed design.

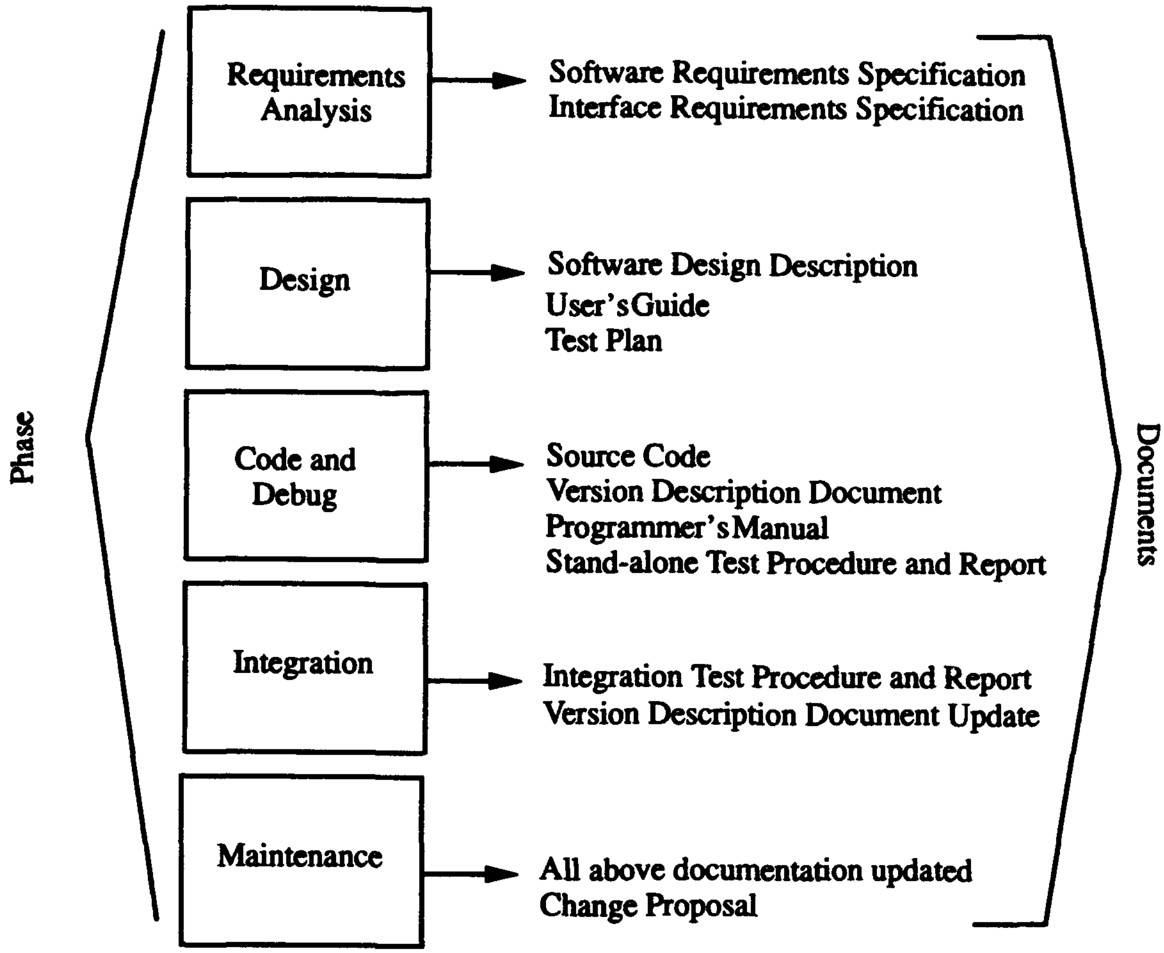

Phase
Requirements Analysis
Design
Code and Debug
Integration
Maintenance
Software Requirements Specification
Interface Requirements Specification
Software Design Description
User's Guide
Test Plan
Source Code
Version Description Document
Programmer's Manual
Stand-alone Test Procedure and Report
Integration Test Procedure and Report
Version Description Document Update
All above documentation updated
Change Proposal
Documents

Fig. I-1 Software Life-Cycle Products.

The main product of the design phase is the software design description documenting all decisions made during the design and identifies the software components constituting the program.

1. **Software Design Description -**

- Top Level Design - Includes hierarchy, data flow, control flow, and a description of the functions performed by high level components (i.e., Computer Software Components (CSCs) per DOD-STD-2167A).

- Detailed Design - Includes a description of the lowest level components in the design (i.e., Computer Software Units (CSUs) per DOD-STD-2167A).

- Requirements Traceability Matrix -Provides a mapping from the requirements to the components of the design (i.e., CSCs and CSUs).

- Data Dictionary - Describes the attributes of program global and local data including valid data ranges, units, update rates, and set/use information for global data.

- Interface Design - Includes the implementation details of the program external interfaces. [2167A DIDs: DI-MCCR-80027 and DI-MCCR-80012]

2. **User's Guide** - This document defines the interfaces to the user that were developed during the design of the software. [2167A DID: DI-MCCR-80019]

3. **Test Plan** - This document describes the software test resources, test requirements, and test schedule for the formal qualification testing of one or more CSCIs (Computer Software Configuration Item). [2167A DID: DI-MCCR-80014A]

I.1.5.3 Code and Debug Phase

During this phase the source code is written and the executable image is generated, debugged and formally tested. The following documents are generated during these activities:

1. **Source Code** - The source code file(s) constitute a document. There must be sufficient comments within the code to make the code readable and understandable. Coding standards define file headers and commenting styles that may vary depending on the type of software and/or the programming language. At a minimum the code headers need to include identification information and a revision history.

2. **Version Description Document (VDD):** The VDD contains information to exactly identify the set of code, data, and documentation constituting a product. Each file that is part of the product is identifiable, at any point in time, by a revision number. When a software product is completed, its version should be identifiable by the set of files constituting the product, with the revision number of each file noted. As the product goes through revisions, the set of files may change and/or the revision numbers for the constituent files may change. The VDD is the place to list the files by revision number for a particular version of the product. For convenience the VDD should also contain a description of the differences between the new version of the software and the previous version. This description might include references to STRs or SCRs.

Although first produced during the code and debug phase, the VDD could be generated when the first document (i.e., software requirements document) is released into the development baseline. It would then be updated each time the development baseline is changed either by the addition of new elements (i.e., documents and source code) or when updates to existing elements are released. It is crucial, though, that the VDD exists at the time the formal testing begins so that the software under test can be referenced exactly.

Some products (i.e., a library model) may be composed of multiple products. In these cases, the VDD identifies the constituent products by name and version numbers. This scheme enables a hierarchical approach to exactly identifying any product or system of products. [2167A DID: DI-MCCR-80013]

3. **Programmer's Manual** - This document contains the instructions to generate and load the executable image of the program from the source code files. Typically, this type of information is composed of compiler and linker/loader command sequences as well as identification of all tools necessary to execute these commands (i.e., identify the computer, operating system, compiler, linker, and loader). The purpose of this document is to allow the maintainer to take a new set of source code and execute it on the target system. [2167A DID: DI-MCCR-80021]

4. **Stand-Alone Test Description** - This document provides a formal description of the tests run on the product to verify that it performs all of its intended functions correctly under normal and stress conditions. Included in this document are the identification of test tools and individual test procedures, an allocation of the product requirements to the test procedures, the actual detailed test procedures, and the test result evaluation instructions. [2167A DIDs: DI-MCCR-80014, DI-MCCR-80015]

5. **Stand-Alone Test Report** - This document provides the actual results of the stand-alone tests with an evaluation of the product performance as measured against the expected results. [2167A DID: DI-MCCR-80017]

I.1.5.4 Integration Phase

During the integration phase, the software developed in the code and debug phase is integrated into its target system. During this activity the following documents are generated:

1. **Integration Test Description** - Similar to stand-alone test description except the scope is for the integration of the product with other software within the system. [2167A DIDs: DI-MCCR-80014, DI-MCCR-80015]

2. **Integration Test Report** - Same as for stand-alone test report except for documenting the results of the integration tests. [2167A DID: DI-MCCR-80017]

3. **Version Description Document** - Update the VDD to include references to the integration test description and report documents. [2167A DID: DI-MCCR-80013]

I.1.5.5 Maintenance Phase

Once a software product has completed the development cycle, it enters the maintenance phase of its life-cycle. During this time changes may be made to the software. These changes are the result of new/changed requirements, planned product improvements, and software "bugs." The following documents are generated during the maintenance activities:

1. **Updates to all other documents** - When the software is updated during this phase, the affected documents will require updates. The above mentioned change proposal should list the documents requiring change (e.g., VDD) and this list should be analyzed for completeness.

2. **Change Proposal** - All changes to released software products require formal documentation that includes a description of the change, reason for the change and the effects of the change. Change Proposals may stem from STRs and SCRs.

I.1.6 Related Policies

DOD-STD-2167A represents the most current government requirements for defense software documentation. The following table lists each of the documents called out by DOD-STD-2167A and how each is addressed by this standard.

Table 6-1 Relationship of DOD-STD-2167A to the CARDS Documentation Std.

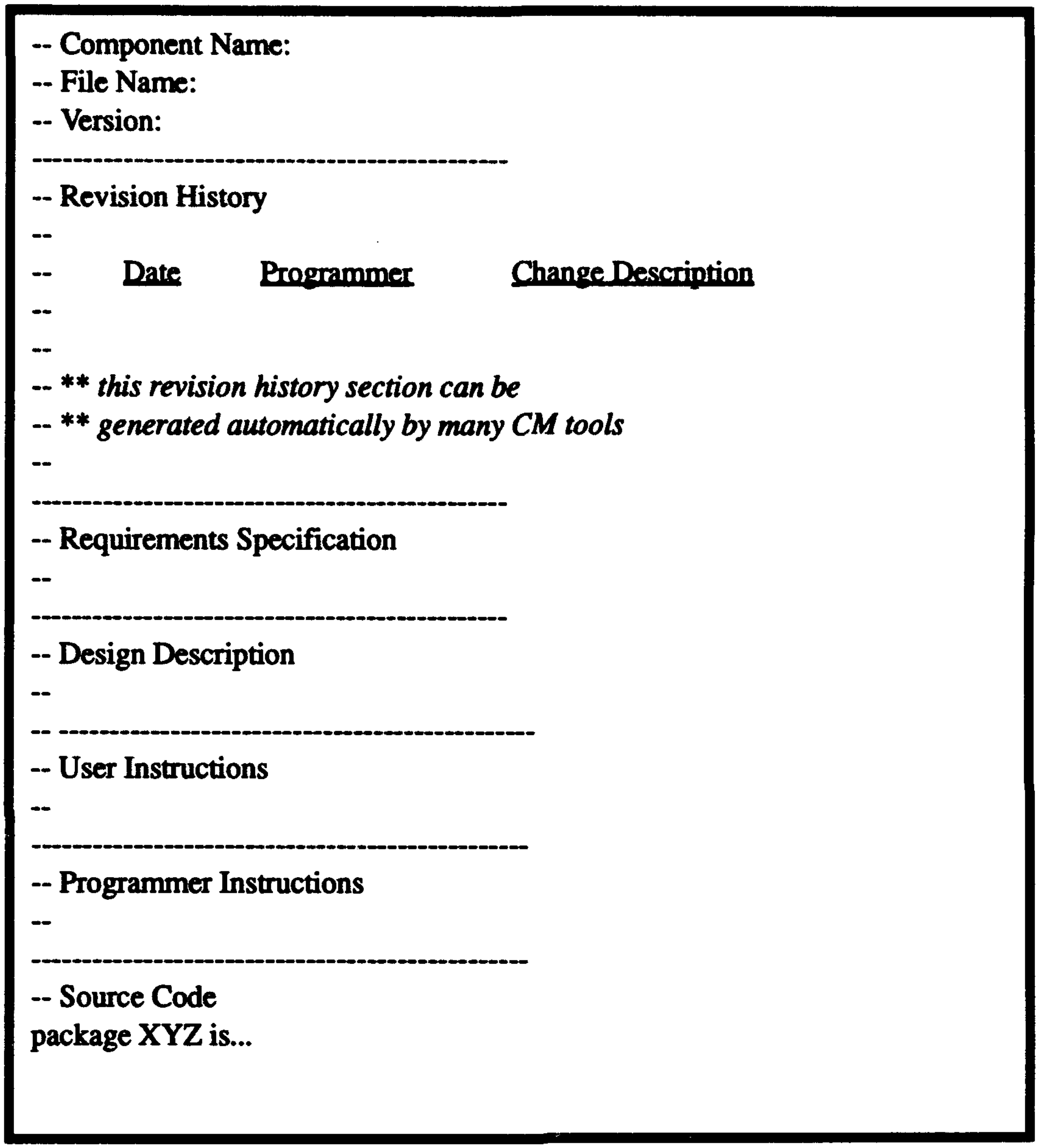
DOD-STD-2167A Document	Applicability To This Standard	Section
System/Segment Design Document	not applicable to CARDS software	-
Software Development Plan	This standard and the LOPP collectively address the applicable requirements of a Software Development Plan	-
Software Requirements Specification	All in-house developed software requires a written specification of its performance and functional requirements.	6.1
Interface Requirements Specification	All in-house developed software requires a written specification of its interface requirements.	6.1
Interface Design Document	Upon completion, the written specification of the interface will also include design constraints.	6.1
Software Design Document	All in-house developed software requires design documentation. Depending on scale, this information may be included in-line with the code.	6.2
Software Product Specification	not applicable to CARDS software	-
Version Description Document	All software will be controlled and identified via its version description documentation.	6.3
Software Test Plan	All software will have documented tests. The documentation will include plans, descriptions, and reports.	6.3, 6.4
Software Test Description	All software will have documented tests. The documentation will include plans, descriptions, and reports.	6.3, 6.4
Software Test Report	All software will have documented tests. The documentation will include plans, descriptions, and reports.	6.3, 6.4

Computer/System Operator's Manual	The LOPP, the User's Manual, and RLF Manuals, and other documents serve as the operator's manuals for the CARDS system. This type of documentation is not applicable for individual software items developed by CARDS	-
Software User's Manual	Each software item produced will provide documentation that addresses the user's perspective.	6.2
Software Programmer's Manual	All software produced for CARDS will contain a description of the procedures and tools needed to modify and build the program.	6.3
Firmware Support Manual	not applicable to CARDS software	-
Computer Resources Integrated Support Document	Future CARDS document will describe the facilities used to develop, test, and maintain the CARDS developed software.	-
Engineering Change Engineering Proposal	Software Trouble Reports act as documentation of changes.	6.5

I.1.7 Example Documentation Templates

I.1.7.1 In-line Ada Documentation Templates

For the case where the software being produced is relatively small (e.g., less than 500 LOC and/or developed by a single person), it may be unreasonable to generate a whole series of documents. The following figure provides an example of how one could provide much of the documentation within the header of an Ada source code file.



```
-- Component Name:
-- File Name:
-- Version:
------------------------------------------
-- Revision History
--
--      Date      Programmer      Change Description
--
--
-- ** this revision history section can be
-- ** generated automatically by many CM tools
--
------------------------------------------
-- Requirements Specification
--
------------------------------------------
-- Design Description
--
-- ------------------------------------------
-- User Instructions
--
------------------------------------------
-- Programmer Instructions
--
------------------------------------------
-- Source Code
package XYZ is...
```

Fig. I-2

I.1.7.2 In-line Script Documentation Template

The following insert is an example of a shell script documentation template.

```
#************************************************
#************************************************
#
# Project Name:
# Script Name:
# Description:
# Type of Script:
# Creation Date:
# Revision Date:
#
#************************************************
#************************************************
#
# $id$
#
#************************************************
#************************************************
#
# Environmental variables used by this script, but
# created outside this script:
#
#************************************************
#************************************************
#
# Variables created and used locally by this script
#
#************************************************
#************************************************
#
#
```

Fig. I-3

I.1.7.3 Version Description Document Template

The following insert is an example Data Item Description for a Version Description Document which provides the high points of the DOD-STD-2167A DID for this document.

1. **Product Identification**

   This section provides the formal name of the software along with its mnemonic and its new version number.

2. **Product Description**

   This section provides a brief description of the product which should include its functions and significant attributes.

3. **Inventory of Elements**

   This section provides a list of each element of the product (e.g., documents, source code files, etc.) by formal name, mnemonic, and fully qualified identification number (e.g., product ID, version number and revision letter).

4. **Change Description**

   This section discusses the changes made to the product since its last release (version). Applicable Change Proposal should be noted (e.g., problem reports).

5. **Known Problems**

   This section lists and describes known defects in the product. Applicable problem reports should be noted.

6. **Planned Changes**

   This section identifies changes that will incorporated into later versions of the product. Applicable Change Proposal should be noted (e.g., engineering change proposals).

7. **Installation Instructions**

   This section describes the sequence of steps necessary to properly install the software

8. **Minimum HW/SW Needed**

   This section describes the minimum hardware and support software needed to run the software.

Fig. I-4

APPENDIX J - Glossary

access authorization number - The number assigned by the CARDS Librarian to a User Account to allow access to CARDS.

accessibility - 1) A measure of openness of a system as determined by policy, network and library connectivity, communications support, and special hardware/software requirements. 2) A measure of extent to which a component facilitates selective use of its parts.

account maintenance - The process of maintaining current user account data through library maintenance procedures as specified by the user's manual.

accountability - A measure of the operational reuse library's capability to collect, relate and utilize audit trails, usage statistics, behavior patterns and other metrics to support enforcement and continuous improvement of its operational policies and procedures.

acquisition - The phase of library population in which potential software products are identified, screened and then evaluated for inclusion in the library.

action - A mechanism permitting a user of the Reuse Library Framework's graphical browser to invoke calls to the underlying operating system, including invoking other tools.

AdaKNET - A semantic network modeling subsystem written in Ada. It provides the heart of the Reuse Library Framework's library model.

AdaTAU - A rule-based inferencing subsystem written in Ada. It supports the AdaKNET semantic network in the Reuse Library Framework's library model.

adaptability - 1) A measure of a library mechanism's capability to represent multiple data models, user defined data models and other user defined tailoring, and the ability to support multi- organizaiion's policies and to incorporate new technology. 2) A measure of the ease with which a component can be altered to fit differing user images and system constraints.

adaptation - The phase of library population in which existing software products are modified or enhancements (i.e., wrappers) are designed, implemented, and tested as new software products.

administrative security - Management rules, procedures, and policies that result in protection of A computer system and its data.

advice - An option offered to a user by the Reuse Library Framework's graphical browser which provides expert guidance by invoking appropriate inferencers to aid the user in navigating a model.

AFS - A distributed, wide-area network file system.

allocation and traceability matrix - A description of the traceability between the domain requirements and the generic architectures.

application - A system which provides a set of general services for solving some type of user problem.

application domain - The knowledge and concepts which pertain to a particular computer application.

architectural constraint - A formalism of the relationships between architectural subsystems and any limitations that may be placed upon them.

architecture-level integration - Combining architecture level components to create a system architecture or domain architecture.

architecture model - A model that represents the interrelationships between system elements and sets a foundation for later requirements analysis and design steps.

architecture modeling - The process of creating the software architecture(s) that implements a solution to the problems in the domain.

architectural requirement - See architectural constraint.

asset - See component.

attribute - A characteristic of an object or relationship. Each attribute has a name and a value.

auditability - A measure of a library's capability to support the capture and analysis of usage statistics, behavior patterns, and other metrics.

authentication - The process of establishing that someone or something is who they say they are.

authorized user - A CARDS library user that has completed the user registration process, has been approved by the CARDS program management, and has been assigned an access authorization number.

browse - Surveying the reusable component descriptions in a library to determine whether the component is applicable to the current application.

CARDS library account registration form (CLARF) - The initial form to be completed by a potential user in order to become an authorized CARDS library user.

cataloging - Placing information about a reusable component into a reusable software library.

certainty - The measure of justifiable assurance or credence that the numeric measure assigned is indeed accurate.

class - A general description of a set of objects that have similarities between their behaviors, attributes, and/or data.

class requirement - Requirements which define the functionality of a specific class.

classification - A mapping of a collection of objects to a taxonomy; the process of determining such a mapping.

classification scheme - The organization of reusable software components according to specific criteria.

closed security environment - An environment in which both the following conditions are true:

1. Application developers have sufficient clearances and authorizations to provide an acceptable presumption that they have not introduced malicious logic.

2. Configuration control provides sufficient assurance that applications and equipment are protected against the introduction of malicious logic prior to and during the operation of system applications.

command center - A facility from which a commander and his/her representatives direct operations and control forces. It is organized to gather, process, analyze, display, and disseminate planning and operational data and to perform other related tasks.

commercial off-the-shelf (COTS) - Commercially available software.

common criteria - Attributes used to evaluate a component regardless of the domain. See component certification.

commonality - Those features which are prevalent in the great majority of applications in a domain.

communications security (COMSEC) - Protection of information while it's being transmitted, particularly via telecommunications. A particular focus of COMSEC is message authenticity.

component - A set of reusable resources that are related by virtue of being the inputs to various stages of the software life cycle, including requirements, design, code, test cases, documentation, etc. Components are the fundamental elements in a reusable software library.

component acquisition - The process of obtaining components appropriate for reuse to be included in a library.

component certification - The process of determining that a component being considered for inclusion in the library meets the requirements of the library and passes all testing procedures. Evaluation takes place against a common set of criteria (reusability, portability, etc.).

component engineer - Responsible for evaluating components for the library, adapting the components if necessary, evaluating component criteria, analyzing the criteria, integrating components, and reporting the findings as appropriate.

component qualification - The process of determining that a potential component is appropriate to the library and meets all quality requirements. Evaluation takes places against domain criteria.

computer security (COMPUSEC) - Protection of information while it's being processed or stored.

concept - An atomic part of the AdaKNET knowledge representation scheme, representing an idea or thing.

Configuration Control Board (CCB) - The authority that is responsible for CARDS configuration management.

configuration management - The purpose of configuration management is to provide version control of any files in the baselined developmental/released library. Configuration management is placed on the files associated with the library model, and component files.

context - The circumstances, situation, or environment in which a particular system exists.

context analysis - The process of defining the extent (or bounds) of a domain for analysis.

context diagram - A top-level data flow diagram showing external interfaces to the process described.

context model - To model the scope of the domain as it exists within a larger domain denoting inputs and outputs.

converged - Said of an AdaKNET role range for which the minimum and maximum have been set to the same value.

customer - The consumer; one who receives our outputs.

data model - A logical representation of a collection of data elements and the association among those data elements.

developer - 1) One who aids in the development of the command center library. 2) One who accesses the CARDS library with the intent of retrieving components for use in developing systems for domain- specific applications.

development configuration manager - The development configuration manager is responsible for overseeing proper configuration management (CM) of developmental hardware and software, library model, library contents, and accompanying documentation. The development configuration manager makes suggestions to the Library Control Board and ensures that decisions of the LCB are implemented. The development configuration manager is not responsible for configuration management of the operational library.

direct extraction - A method of component retrieval in which the component is retrieved directly from a library.

discretionary access control (DAC) - An access policy that restricts access to system objects (e.g., files, directories, devices) based on the identity of the users and/or groups to which they belong. "Discretionary" means that a user with certain access permissions is capable of passing those permissions to another user (e.g., letting another user modify a file).

domain - An area of activity or knowledge containing applications which share a set of common capabilities and data.

domain analysis - The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

domain analyst - An individual skilled in domain analysis methodologies. The Domain Analyst is responsible for defining the language, tools, and techniques used in performing the domain analysis. This person also documents the domain model and may be responsible for defining any generic architectures associated with the domain.

domain architecture - High-level paradigms and constraints characterizing the commonality and variances of the interactions and relationships between applications within a domain.

domain constraints - Represent the mission-level requirements identified within the boundaries of the domain. They determine the functionality of the system expressed in terms and language dominant within the domain.

domain criteria - Specifications a potential component must adhere to in order to obtain acceptability in the domain and inclusion in the library. Domain criteria are a composite of three sets of constraints: component constraints, architectural constrains, and implementation constraints.

domain engineering - An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools, and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.

domain entity - An object, relationship, operator, process, fact, or rule of a domain.

domain expert - An individual with extensive knowledge of a particular domain.

domain language - A collection of rules which relate objects and functions and which can be explicit and be encapsulated in a formal language; further, it can be used to specify the construction of systems in the domain.

domain-level integration - The process of using and evolving domain and application components in the creation of requirements, architectures and implementations (domain and application).

domain model - A definition of the functions, objects, data, and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions.

domain modeling - The process of encoding knowledge about a domain into a formalism.

domain requirement - See domain constraint.

domain-specific library - A library whose components are bound by a specific domain.

domain-specific reuse - Reuse that is targeted for a specific domain (as opposed to reuse of general purpose workproducts). It typically involves reuse of larger workproducts (subsystems, architectures, etc.) that general purpose reuse.

domain-specific software architecture - An architecture (interactions and relationships between objects) used to develop software applications based on a specific domain.

encode - See library encoding. entity - A particular and discrete unit; a named product, process object, or relationship.

expandability - The extent to which a library or component allows for adding new components or functions.

extensibility - 1) A measure of the ability to modify and enhance the operational reuse library's data model and contents while minimizing interruption to subscribers. 2) The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs.

extraction - See retrieval.

facet - A perspective, viewpoint, or dimension of a particular domain which can be represented as a class.

feature - A prominent or distinctive user-detectable aspect, quality, or characteristic of a software system or systems.

field site - The location of an organization utilizes Command Center Library or installs an instance of a CARDS-based domain-specific library.

file security - Protection of files stored on a computer system through discretionary access control and/or mandatory access control.

fill - An AdaKNET term for an individual that is used as the type of an aggregation role emanating from another individual. (Also called satisfy.)

flexibility - 1) A measure of the operational reuse libraries' ability to accommodate changing subscriber requirements, such as handling of proprietary software, organization's policies and new technology. 2) The extent to which a component's missions, functions, or data satisfy other requirements. franchise - an instance of a domain-specific infrastructure built utilizing the CARDS Concept of Operations/Franchise Plan.

franchisee - Group to whom a franchise is granted.

generic architecture - A collection of high-level paradigms and constraints that characterize the commonality and variances of the interactions and relationships between the various components in a system.

generic command center architecture - The fundamental generic architecture that underlies command center applications.

government off-the-shelf (GOTS) - Software developed for and owned by the government.

graphical browser - A graphical presentation of the domain model and interrelations between components. Through the graphical browser, components may be browsed, viewed, and extracted. It also provides an inferencing mechanism to aid in prototyping and selecting the correct components.

horizontal domain - The knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application domain.

identified component list - A list of components that (partially) satisfied each subsystem identified within the generic architecture.

implementation - level integration - Combining components in order to implement a system.

implementation constraint - Provides the hardware and software requirements to which the individual software modules must adhere.

implementation requirement - See implementation constraint.

indirect extraction - A component retrieval method in which a component is retrieved from a remote library with interactive extraction. This can occur transparently or non- transparently.

individual - Used in AdaKNET to represent an actual instance of a concept; generally it is something that exists in the world in some manner.

individuation - In AdaKNET, the act of declaring or creating an individual.

inferencer - A mechanism which uses existing facts and rules about the elements of a model to perform reasoning about that model and deduce new facts and rules.

information security (INFOSEC) - Protection of Information.

infrastructure - The basic underlying framework or features.

inheritance - A mechanism whereby classes make use of the procedures, attributes, and/or data defined in other classes.

integration - The process (in library population) of verifying that a software product meets the architectural constraints imposed by the generic architecture.

interoperability - The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

knowledge blueprint - A flexible plan to transition knowledge to the community.

knowledge engineer - An individual responsible for modeling domains. The knowledge engineer works closely with the domain analyst and the domain expert in encoding the domain analysis products in to a library model.

knowledge representation - Codification of domain knowledge.

librarian - CARDS library cataloger/maintainer.

library - A collection of components that are cataloged according to a common classification scheme and a set of applications that provide a mechanism to browse and retrieve components.

library account holder - a user or staff member who is authorized to access the CARDS library. There are currently five account types whicł an account holder may be assigned to a library account holder:

1. Staff Developer Account - holders of this account type have a typical Unix account and an AFS account with RLF development directory privileges.

2. Staff Account - holders of this account type have a typical Unix and an AFS account with restrictions in RLF development directories.

3. Staff Affiliate Account - holders of this account type have a typical Unix account and an AFS account with restrictions in RLF development directories and the living documents directory.

4. Users with Sun4 and AFS - holders of this account type will have an AFS account only, with restrictions in RLF development directories and in the living documents directory.

5. Users without AFS or without Sun4 - holders of this account type have a restricted Unix account and an AFS account with restrictions in RLF development directories and in the living documents directory.

library applications - Services provided to the library user.

library asset node - CARDS/RLF graphical browser's representation of a component.

Library Control Board (LCB) - The deliberative body which controls the decision-making process for making recommendations about library components (e.g., which components to acquire, reject, or modify). The LCB also has approval authority for determining when to release new versions of a library to the Configuration Control Board for customer release. The LCB is concerned with the technical issues related to library construction.

library encoding - The process of encoding the products of the domain analysis into a library model.

library model - A model that represents the domain components and the relationships between them.

library population - The process of acquiring/developing components in support of the library model.

life cycle - All the activities (e.g., design, code, test) a component is subjected to from its inception until it is no longer useful. A life cycle may be modeled in terms of phases, which are often characterizations of activities by their purpose or function such as design, code, or test.

life-cycle artifact - A product of the software engineering process (i.e., a component).

mandatory access control (MAC) - An access policy that restricts access to system objects (e.g., files, directories, devices) based on the sensitivity of the information in the object (represented by the object's label) and the authorization of the subject (usually represented by the user's clearance) to access the information at that sensitivity level. "Mandatory" means that the system enforces the policy; users do not have discretion to share their files.

memorandum of understanding - An agreement stating terms of cooperation between two entities.

metrics - Quantitative and qualitative analysis values calculated and collected according to a precise definition ana used to establish comparative aspects of development progress, quality assessment, or choice of options.

model - A representation of a real-world process, device, or concept.

modeling - The process of creating a model.

multilevel security - Information processing and communications which allows two or more security levels of information to be processed simultaneously within the same system when some users are not cleared for all levels of information present.

name (role) - The name of an AdaKNET aggregation relationship.

object - A persistent data item in a library. Each object may be characterized by behavior (procedures), attributes (system and user-defined) and/or the data required to carry out the behavior.

operability - A measure of the ease of learning versus ease of use of the library mechanism's capability to support searches, retrievals, extractions and contributions.

physical security - Protection of the physical computer system and related buildings and equipment from fire and other natural and environmental hazards, as well as from intrusion. Also covers the use of locks, keys, and administrative measures used to control access to computer systems and facilities.

procedural security - See administrative security.

prototyping - The practice of building a first or original model (sometime scaled down, but accurate) of a system to verify the operational process prior to building a final system.

quality assurance - Quality assurance of the products generated by the library development process is built into the process as part of Total Quality Management (TQM). The LCB ensures the quality of the products in the developmental library. The role of quality assurance is also to monitor the adherence of the library developers to the library development processes.

RLF - Reuse Library Framework. Provides a framework for building domain-specific libraries.

range (role) - The number of simultaneous copies that may exist of an AdaKNET aggregation relationship.

rapid prototyping - The process of using a library mechanism to quickly prototype a system.

reengineering - The process of examining, altering, and re-implementing an existing computer system to reconstitute it in a new form. It uses the practices such as restructuring, reverse engineering, and migration to identify and separate those systems worth maintaining from those that should be replaced; to extend the useful life of existing systems; and to perform maintenance more efficiently and correctly.

refinement - Specialization of a category; e.g., Ada software is a refinement of software.

relationships - The connections between entities, objects, or components.

repository - The mechanism for defining, storing, and managing all information, concerning an enterprise and its software systems - logical data and process models, physical definitions and code, and organization models and business rules.

retrieval - The process of obtaining a component from a library such that it may be used in the development process.

reusable component - A component (including requirements, designs, code, test data, specifications, documentation, expertise, etc.) designed and implemented for the specific purpose of being reused.

reuse - The application of existing solutions to the problems of system development. Reuse involves transfer of expertise encoded in software-related work products. The simplest form of reuse from software work products is the use of subroutine/subprogram libraries for string manipulations or mathematical calculations.

reuse library - A library specifically designed, built, and maintained to house reusable components.

reuser - One who implements a system through the reuse process.

reverse engineering - The process of analyzing a computer system to identify its components and their interrelationships.

role - An AdaKNET term referring to an aggregation ("consists of") relationship between two concepts. See name, range, and type.

role restriction - An inherited AdaKNET aggregation relationship which is narrowed at the inheriting concept, by further restricting the range or type.

rule base - A collection of rules about the elements of a domain. A rule describes when and how the facts about the model may change.

Rule Base Definition Language (RBDL) - An application specific language (ASL) used to define AdaTAU inference bases.

satisfy - An AdaKNET term which is said of an individual that is used as the type of an aggregation role emanating from another individual. (Also called fill).

scaleability - A measure of number, diversity, and size of components that can be managed by a library mechanism.

schema - The data describing the context and meaning of system information.

security model - A precise statement of the security rules of a system.

security policy - The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

security testing - A type of testing in which testers determine whether the security features of a system are implemented as designed. Security testing may include hands-on functional testing, penetration testing, and formal verification.

semantic network - A graphical knowledge representation method composed of nodes linked to each other.

Semantic Network Definition Language (SNDL) - An application specific language (ASL) used to define AdaKNET semantic network models.

software architecture - High-level paradigms and constraints characterizing the structure of operations and objects, their interfaces, and control to support the implementation of applications in a domain. Includes the description of each software component's functionality, name, parameters and their types and a description of the component's interrelationships.

software engineering environment (SEE) - The computer hardware, operating system, tools, computer- hosted capabilities, rules, and techniques that assist in the development and production of software.

specialization - The act of declaring that one concept represents a narrowing of the idea represented by another concept.

standard descriptors - Basic conceptual units forming the interface between domain architectures and reusable components; i.e., high-level mini-specs for a class of components.

subsystem - Conceptual aggregate of complimentary functions within an architecture.

surrogate retrieval - A user's library retrieves a component from a remote library for the user.

system architecture - A model that represents the interrelationship between system elements and sets a foundation for later requirements analysis and design steps.

system composition - The automatic configuration of a prototype system based on hardware and software requirements.

system engineering - A process encompassing requirements gathering at the system level with a small amount of top-level design and analysis.

taxonomy - The theory, principles, and process of categorizing entities in established categories.

technical reference model - A conceptual description of the functionalities encompassed within the domain.

tree - A data structure consisting of nodes interconnected by arcs depicting relationships between the nodes.

type (role) - The allowable range of values of an AdaKNET aggregation role.

usage statistics - Measured and collected statistics evaluating the overall performance of the library mechanism; these include information about the library's users, system efficiency, system reliability and summary of queries.

user - An individual assigned an access authorization number and authorized to access the CARDS library to develop systems for domain-specific applications.

user account - The physical account established for each user containing pertinent information relative to that user.

user account log - A record of information pertaining to the stages of the enrollment and deactivation process for each user.

user processing support - Information processing allowing the user to tailor functions of the domain software to address particular needs.

user registration - The process of enrolling a potential CARDS library user.

user support - All functions related to the technical and administrative support of the end user.

vertical domain - The knowledge and concepts that pertain to a particular application domain.

APPENDIX K - References

1. Central Archive for Reusable Defense Software. *Acquisition Handbook - Update.* Informal Technical Report UPDATE – STARS-AC-04105/001/. Paramax Systems Corporation. 30 April 1993.

2. Central Archive for Reusable Defense Software. *Command Center Library Model Document.* Informal Technical Report STARS-AC-04119. Paramax Systems Corporation. 15 July 1993.

3. Central Archive for Reusable Defense Software. *Franchise Plan.* Informal Technical Report STARS-AC-04116/000/00. Paramax Systems Corporation. 30 March 1993.

4. Cohen, S. G., et al, "Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain" *Technical Report, CMU/SEI-91-TR-28, ESD-91-TR-28,* June 1992.

5. DISA Domain Analysis Guidelines, *draft*, 1992, DoD Software Reuse Initiative.

6. *Library Operation Policies and Procedures for the Central Archive for Reusable Defense Software: Volume I and II.* Informal Technical Reports STARS-AC-B004/001/02 & STARS-AC-B004/003/01. Paramax Systems Corporation. 30 September 1993.

7. *Organization Domain Modeling (ODM), Volume I - Conceptual Foundations, Process and Workproduct Descriptions Version 1.0.* Informal Technical Reports STARS-UC-15156/ 024/00. Paramax Systems Corporation. 29 June 1993.

8. Wartik, S., Prieto-Diaz, R., "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches" *The International journal of Software Engineering and Knowledge Engineering*, September 1992.

9. CECOM, "Impact of Domain Analysis on Reuse Methods" *Final Report CIN: C04-087LD-0001-00,* 6 November 1989.